Michel Abdalla
David Pointcheval
Pierre-Alain Fouque
Damien Vergnaud (Eds.)

LNCS 5536

Applied Cryptography and Network Security

7th International Conference, ACNS 2009
Paris-Rocquencourt, France, June 2009
Proceedings

Springer

Lecture Notes in Computer Science 5536

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

Michel Abdalla David Pointcheval
Pierre-Alain Fouque Damien Vergnaud (Eds.)

Applied Cryptography and Network Security

7th International Conference, ACNS 2009
Paris-Rocquencourt, France, June 2-5, 2009
Proceedings

Springer

Volume Editors

Michel Abdalla
David Pointcheval
Pierre-Alain Fouque
Damien Vergnaud
École Normale Supérieure
45, rue d'Ulm, 75230 Paris Cedex 05, France
E-mail: {michel.abdalla, david.pointcheval,
pierre-alain.fouque, damien.vergnaud}@ens.fr

Library of Congress Control Number: Applied for

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

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-642-01956-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-01956-2 Springer Berlin Heidelberg New York

springer.com

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12683758 06/3180 5 4 3 2 1 0

Preface

ACNS 2009, the 7th International Conference on Applied Cryptography and Network Security, was held in Paris-Rocquencourt, France, June 2–5, 2009. ACNS 2009 was organized by the École Normale Supérieure (ENS), the French National Center for Scientific Research (CNRS), and the French National Institute for Research in Computer Science and Control (INRIA), in cooperation with the International Association for Cryptologic Research (IACR). The General Chairs of the conference were Pierre-Alain Fouque and Damien Vergnaud.

The conference received 150 submissions and each submission was assigned to at least three committee members. Submissions co-authored by members of the Program Committee were assigned to at least four committee members. Due to the large number of high-quality submissions, the review process was challenging and we are deeply grateful to the committee members and the external reviewers for their outstanding work. After meticulous deliberation, the Program Committee, which was chaired by Michel Abdalla and David Pointcheval, selected 32 submissions for presentation in the academic track and these are the articles that are included in this volume. Additionally, a few other submissions were selected for presentation in the non-archival industrial track. The best student paper was awarded to Ayman Jarrous for his paper "Secure Hamming Distance Based Computation and Its Applications," co-authored with Benny Pinkas. The review process was run using the iChair software, written by Thomas Baigneres and Matthieu Finiasz from EPFL, LASEC, Switzerland and we are indebted to them for letting us use their software.

The program also included four invited talks in addition to the academic and industrial tracks. The invited talks were given by Craig Gentry from Stanford University on "Fully Homomorphic Encryption Using Ideal Lattices," Antoine Joux from DGA and the University of Versailles on "Can We Settle Cryptography's Hash?," Angelos Keromytis from Columbia University on "Voice Over IP: Risks, Threats and Vulnerabilities," and Mike Reiter from the University of North Carolina at Chapel Hill on "Better Architectures and New Applications for Coarse Network Monitoring." We would like to genuinely thank them for accepting our invitation and for contributing to the success of ACNS 2009.

Finally, we would like to thank our sponsors Ingenico, CNRS, and the French National Research Agency (ANR) for their financial support and all the people involved in the organization of this conference. In particular, we would like to thank the Office for Courses and Colloquiums (*Bureau des Cours-Colloques*) from INRIA and Gaëlle Dorkeld for their diligent work and for making this conference possible.

June 2009

Michel Abdalla
David Pointcheval
Pierre-Alain Fouque
Damien Vergnaud

ACNS 2009

7th Annual Conference on
Applied Cryptography and Network Security

Paris-Rocquencourt, France
June 2–5, 2009

Organized by

École Normale Supérieure (ENS)
Centre National de la Recherche Scientifique (CNRS)
Institut National de Recherche en Informatique et en Automatique (INRIA)

In Cooperation with
The International Association for Cryptologic Research (IACR)

General Chairs

Pierre-Alain Fouque	École Normale Supérieure, France
Damien Vergnaud	École Normale Supérieure, France

Program Chairs

Michel Abdalla	École Normale Supérieure, France
David Pointcheval	École Normale Supérieure, France

Program Committee

Gildas Avoine	Université Catholique de Louvain, Belgium
Feng Bao	Institute for Infocomm Research, Singapore
Christophe Bidan	Supélec, France
Alex Biryukov	University of Luxembourg
Xavier Boyen	Stanford University, USA
Dario Catalano	University of Catania, Italy
Liqun Chen	Hewlett Packard Labs, UK
Jean-Sébastien Coron	University of Luxembourg
Jacques Demerjian	CS, France
Aline Gouget	Gemalto, France
Louis Granboulan	EADS, France
Peter Gutmann	University of Auckland, New Zealand
Nick Howgrave-Graham	NTRU Cryptosystems, USA
Stanislaw Jarecki	University of California at Irvine, USA
Marc Joye	Thomson R&D, France
Jaeyeon Jung	Intel, USA

Seny Kamara	Microsoft Research, USA
Jonathan Katz	University of Maryland, USA
Aggelos Kiayias	University of Connecticut, USA
Xuejia Lai	SJTU, China
Javier Lopez	University of Malaga, Spain
Olivier Orcière	Thales, France
Kenny Paterson	Royal Holloway, University of London, UK
Giuseppe Persiano	University of Salerno, Italy
Josef Pieprzyk	University of Macquarie, Australia
Matt Robshaw	Orange Labs, France
Kazue Sako	NEC, Japan
Palash Sarkar	Indian Statistical Institute, India
Berry Schoenmakers	TUE, The Netherlands
Hovav Shacham	University of California at San Diego, USA
Jessica Staddon	PARC, USA
Michael Szydlo	Akamai, USA
Serge Vaudenay	EPFL, Switzerland
Avishai Wool	Tel Aviv University, Israel
Duncan Wong	City University of Hong Kong
Jianying Zhou	Institute for Infocomm Research, Singapore

Steering Committee

Yongfei Han	ONETS, China
Moti Yung	Google, USA
Jianying Zhou	Institute for Infocomm Research, Singapore

External Reviewers

Asmaa Adnane
Toshinori Araki
Joonsang Baek
Aurélie Bauer
Bruno Blanchet
Carlo Blundo
Emmanuel Bresson
Sébastien Canard
Ran Canetti
Richard Chow
Pascal Delaunay
Valeria de Paiva
Mario Di Raimondo
Ming Duan
Renaud Dubois
Dario Fiore
Guillaume Fumaroli
Jun Furukawa
Martin Gagne
Clemente Galdi
David Galindo
Benedikt Gierlichs
Jens Groth
Gilles Guette
Sylvain Guilley
Wei Han
Javier Herranz
Duong Hieu Phan
Tsz Hon Yuen
Qiong Huang
Emeline Hufschmitt
Vincenzo Iovino
Toshiyuki Isshiki
Amandine Jambert
Haimin Jin
Pascal Junod
Mohamed Karroumi
Dmitry Khovratovich
Chung Ki Li
Eike Kiltz
Ilya Kizhvatov
Hugo Krawczyk
Miroslaw Kutylowski
Sylvain Lachartre
Cédric Lauradoux
David Lefranc
Francois Lesueur
Tieyan Li

Wei Li
Joseph K. Liu
Yu Long
Xianhui Lu
Subhamoy Maitra
Krzysztof Majcher
Mark Manulis
Sandra Marcello
Tania Martin
Krystian Matusiewicz
Petros Mol
Jorge Nakahara Jr
Yossi Oren
Khaled Ouafi
Pascal Paillier
Philippe Painchault
Sylvain Pasini
Maura Paterson
Serdar Pehlivanoglu
Kun Peng
Duong Hieu Phan
Gilles Piret
Nicolas Prigent
Sasa Radomirovic
Louis Salvail
Koby Scheuer
Roman Schlegel
Yannick Seurin
Elaine Shi
Igor Shparlinski
Vladimir Shpilrain
Hervé Sibert
François-Xavier Standaert
Ron Steinfeld
Xiaorui Sun
Christophe Tartary
Isamu Teranishi
Frederic Tronel
Ivan Visconti
Zhongmai Wan
Mi Wen
Jian Weng
Douglas Wikström
Charles Wright
Hongjun Wu
Yongdong Wu
Yaying Xiao
Guomin Yang
Yanjiang Yang
Yang Yanjiang
Bin Zhang
Hong-Sheng Zhou
Huafei Zhu

Sponsoring Institutions

Ingenico, Neuilly-sur-Seine, France
The French National Research Agency (ANR), Paris, France
French National Center for Scientific Research (CNRS), Paris, France

Table of Contents

Key Exchange

Secure Computation

Public-Key Encryption

Network Security I

Traitor Tracing

Authentication and Anonymity

Hash Functions

Network Security II

Lattices

Side-Channel Attacks

Group Key Exchange Enabling On-Demand Derivation of Peer-to-Peer Keys

Mark Manulis

Cryptographic Protocols Group
Department of Computer Science
TU Darmstadt & CASED, Germany
mark@manulis.eu

Abstract. We enrich the classical notion of group key exchange (GKE) protocols by a new property that allows each pair of users to derive an independent *peer-to-peer (p2p) key* on-demand and without any subsequent communication; this, in addition to the classical *group key* shared amongst all the users. We show that GKE protocols enriched in this way impose new security challenges concerning the secrecy and independence of both key types. The special attention should be paid to possible collusion attacks aiming to break the secrecy of p2p keys possibly established between any two non-colluding users.

In our constructions we utilize the well-known parallel Diffie-Hellman key exchange (PDHKE) technique in which each party uses the same exponent for the computation of p2p keys with its peers. First, we consider PDHKE in GKE protocols where parties securely transport their secrets for the establishment of the group key. For this we use an efficient multi-recipient ElGamal encryption scheme. Further, based on PDHKE we design a generic compiler for GKE protocols that extend the classical Diffie-Hellman method. Finally, we investigate possible optimizations of these protocols allowing parties to *re-use* their exponents to compute both group and p2p keys, and show that not all such GKE protocols can be optimized.

1 Introduction

Traditional *group key exchange (GKE)* protocols allow users to agree on a secret *group key* and are fundamental for securing applications that require group communication. However, messages authenticated or encrypted with the group key attest only that the originator of the message is a valid member of the group. The goal of this paper is to investigate the enrichment of GKE protocols with the additional derivation of *peer-to-peer (p2p) keys* for any pair of users. A single run of a GKE protocol enriched in this way would suffice to set up a secure group channel providing possibly each pair of users with an independent secure peer-to-peer channel "for free", thus implicitly allowing for a secure combination of group and p2p communication. Note that messages authenticated or encrypted with a p2p key would attest not only the group membership but also allow for the identification of the sender. For example, in digital conferences or instant messaging systems each user can participate in a secure group discussion and if necessary switch for a while to a secure bilateral discussion with some other user; or a

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 1–19, 2009.

user can encrypt some file for all users using the group key and attach supplementary files encrypted with p2p keys for the selected subset of its peers.

Obviously, the simultaneous computation of group and p2p keys can be achieved through the execution of a GKE protocol in parallel with the execution of a two-party key exchange (2KE) protocol between every pair of users. The drawback of this approach is that it would require $(n^2 - n)/2$ parallel 2KE executions in order to provide each pair with the own key (where n is the number of users). The only way to avoid such parallel 2KE executions is to consider solutions where p2p keys are computed *on-demand*; we denote such GKE protocols by GKE+P.

A rather naïve construction of GKE+P protocols can be obtained from the execution of a GKE protocol followed by a separate execution of a 2KE protocol between some pair of users. The drawback of this solution is the additional interaction for the computation of p2p keys (in the worst case requiring up to $n - 1$ different 2KE protocol runs involving the same user) and the deployment of two different protocols (GKE and 2KE). Therefore, since GKE participants already interact to establish the group key it appears interesting to investigate whether GKE+P protocols can be constructed enabling the completely *non-interactive* derivation of p2p keys?

GKE+P protocols raise new security challenges concerning the independence of group and p2p keys. Traditional GKE protocols require that a session group key remains secret from any adversary that is an external entity to that session. In GKE+P protocols this requirement should hold even in case where p2p keys leak. By the same token GKE+P protocols should provide secrecy of the p2p keys computed in some session independent of whether the adversary learns the group key or not. However, the most significant challenge specific to GKE+P protocols results from the independence amongst different p2p keys computed in the same session and even by the same user (for different peers). In particular GKE+P protocols should provide secrecy of some session p2p key if other participants that are not intended to compute that key collude. Thus, when defining the secrecy of some session p2p key we should no longer assume that the adversary remains an external entity to that session but rather that it may act on behalf of colluding participants and thus deviate from the protocol specification.

Specification of the appropriate security requirements and efficient, provably secure solutions for GKE+P protocols represents the main focus of our work.

1.1 Related Work

The basic security goal of any key exchange protocol is called (Authenticated) Key Exchange security ((A)KE-security, for short) and deals with the secrecy or indistinguishability of the established session group key with respect to an (active) adversary which is usually modeled as an external entity from the perspective of the attacked session. This requirement became an inherent part of all security models for 2KE protocols, e.g. [3,5,6,7,17,18,19,34,38], and GKE protocols, e.g. [10,11,13,15,28,29]. A general signature-based compilation technique proposed by Katz and Yung [29] can turn any KE-secure (group) key exchange protocol into an AKE-secure one, thus by adding the authentication and thwarting possible impersonation attacks. Additionally, we remark that some of the mentioned security models for GKE protocols (e.g. [12, 13, 28]) aim at defining optional security against insider attacks, and the corresponding compilers defined in these papers can turn any AKE-secure GKE protocol into a protocol that withstands such attacks. These compilers also provide the so-called requirement of mutual authentication (MA) [7, 11, 15], which ensures the bilateral authentication of all protocol participants and is usually combined with a key confirmation step.

From the variety of the existing GKE protocols (see [9, 35] for surveys) of special interest in the context of our GKE+P constructions are the (unauthenticated) extensions of the classical 2KE approach by Diffie and Hellman [21] to a group setting, e.g. [16, 20, 24, 31, 32, 37, 39, 40]. Let us denote all these protocols for simplicity as *Group Diffie-Hellman (GroupDH)* protocols since they derive the group key from some shared secret which in turn depends on the individual exponents chosen by the protocol participants during the execution. For the design of GKE+P protocols it appears promising to investigate to what extent the existing GroupDH protocols allow for the non-interactive, on-demand computation of p2p keys, in particular whether or not secret exponents used in these GroupDH protocols can be safely re-used for the computation of p2p keys.

GKE protocols proposed in [1, 36] are partially related since they consider a 2KE protocol as a building block in order to obtain a secure GKE protocol, yet without enabling on-demand computation of p2p keys amongst any pair of users. Also, the so-called *group secret handshakes* [25, 26] should be noticed since these can be seen as extensions of GKE protocols with another property called affiliation-hiding. We mention them here since the on-demand computation of p2p keys can be also considered in that scenarios (in particular our results can be extended to deal with [25] that is based on the GKE protocol from [16]).

One of the main building blocks across all our GKE+P constructions is the *parallel* execution of the 2KE Diffie-Hellman protocol (PDHKE), in which each user broadcasts a value of the form g^x (for the appropriate generator g and private user's exponent x) and uses x for the computation of different p2p keys. In this context, Jeong and Lee [27] recently specified and analyzed a related mechanism where keys are derived in parallel from ephemeral and long-lived exponents. However, their security model does not consider collusion attacks against the secrecy of p2p keys computed by non-colluding users. Note also the recent work by Biswas [8] who revised the 2KE Diffie-Hellman protocol allowing its participants to choose two different exponents each and obtain 15 different shared keys.

1.2 Contributions and Organization

We start in Section 2 with the extension of the classical GKE security model from [29] in order to address the additional challenges of GKE+P protocols and define the corresponding requirements of (A)KE-security of group and p2p keys; the latter in the presence of collusion attacks. Our model is designed in a modular way and can be selectively applied to GKE+P and GKE protocols, and also to the protocols like PDHKE. In Section 3 we introduce general notations and recall some classical assumptions.

In Section 4 we present and analyze our first GKE+P protocol, denoted PDHKE-MRE. In this protocol we merge PDHKE with the multi-recipient ElGamal encryption (MRE) from [4, 33]. PDHKE-MRE optimizes the combination of PDHKE and MRE in that it utilizes user's exponent for both — generation of p2p keys and decryption of

ElGamal ciphertexts. This optimization is tricky (compared to the simple "black-box" combination) since it requires an additional hardness assumption. Our security analysis of PDHKE-MRE also demonstrates that PDHKE can be used as a stand-alone protocol to obtain KE-secure p2p keys in the presence of collusion attacks.

In Section 5 we obtain more efficient GKE+P protocols from GroupDH protocols (see related work for examples). First, we describe a general compilation technique to obtain GKE+P solutions from any GroupDH protocol based on PDHKE, yet assuming that the exponents used for the derivation of p2p keys are independent from those used in the computation of the group key. Additionally, we investigate whether private exponents that are implicit to the GroupDH protocols can be re-used for the on-demand computation of p2p keys. The key observation here is that many GroupDH protocols require each user U_i to choose some exponent x_i and broadcast a public value g^{x_i}. The natural question is whether a value $g^{x_i x_j}$, if computed from the exponents x_i and x_j used in the GroupDH protocol, would be suitable for the derivation of a secure p2p key between U_i and U_j? In this light we analyze the well-known communication-efficient protocols by Burmester and Desmedt (BD) [16] and by Kim, Perrig, and Tsudik (KPT) [31] (the latter as a representative for the family of Tree Diffie-Hellman protocols). We show that in the BD protocol this technique will not guarantee the KE-security of p2p keys, whereas in the KPT protocol it will, though at the cost of an additional hardness assumption. The latter result is of special interest since we do not introduce any new communication costs to the KPT protocol.

In Section 6, we compare the performance of the introduced GKE+P protocols.

In Section 7 we show that the authentication compiler introduced in [29] for securing traditional KE-secure GKE protocols is also sufficient for adding the authentication to KE-secure GKE+P protocols.

2 Security Model for GKE+P Protocols

Our security model for GKE+P protocols extends the meanwhile standard GKE security model from [29] by capturing the additional requirements concerning the on-demand computation of p2p keys.

2.1 Participants, Sessions, and Correctness of GKE+P Protocols

By $\mathcal{U}$ we denote a set of at most N users (more precisely, their identities which are assumed to be unique) in the universe. Any subset of n users ($2 \leq n \leq N$) can participate in a single session of a GKE+P protocol $\mathcal{P}$. Each $U_i \in \mathcal{U}$ holds a (secret) long-lived key LL_i.[1] The participation of U_i in distinct, possibly concurrent protocol sessions is modeled via an unlimited number of *instances* Π_i^s, $s \in \mathbb{N}$. Each instance Π_i^s can be invoked for one session with some partner id $\mathtt{pid}_i^s \subseteq \mathcal{U}$ encompassing the identities of the intended participants (including U_i). At the end of the interactive phase Π_i^s holds

[1] Our GKE+P protocols are first analyzed in the authenticated links model where long-lived keys are assumed to be empty. The authentication in GKE+P protocols using the compiler technique from [29] that we discuss in Section 7 will assume that each LL_i corresponds to some digital signature key pair.

a *session id* $\mathtt{sid}_i^s$ which uniquely identifies the session. Two instances Π_i^s and Π_j^t are considered as *partnered* if $\mathtt{sid}_i^s = \mathtt{sid}_j^t$ and $\mathtt{pid}_i^s = \mathtt{pid}_j^t$. The success of the interactive phase by some instance Π_i^s is modeled through its *acceptance*, in which case the instance holds a *session group key* k_i^s. Each instance Π_i^s that has accepted can later decide to compute a *session p2p key* $k_{i,j}^s$ for some user $U_j \in \mathtt{pid}_i^s$. We are now ready to formally define what a GKE+P protocol is.

Definition 1 (GKE+P Protocol and Correctness). *$\mathcal{P}$ is a* group key exchange protocol enabling on-demand derivation of p2p keys (GKE+P) *if $\mathcal{P}$ consists of the group key exchange protocol* GKE *and a p2p key derivation algorithm* P2P *defined as follows:*

$\mathcal{P}$.GKE$(U_1, \ldots, U_n)$*: For each input U_i a new instance Π_i^s is created and a probabilistic interactive protocol between these instances is executed such that at the end every instance Π_i^s accepts holding the session group key k_i^s.*

$\mathcal{P}$.P2P(Π_i^s, U_j)*: On input an accepted instance Π_i^s and some user identity $U_j \in \mathtt{pid}_i^s$ this deterministic algorithm outputs the session p2p key $k_{i,j}^s$. (We assume that* P2P *is given only for groups of size $n \geq 3$ since for $n = 2$ the group key is sufficient.)*

A GKE+P protocol $\mathcal{P}$ is correct *if (when no adversary is present) all instances participating in the protocol $\mathcal{P}$.*GKE *accept with identical group keys and $\mathcal{P}$.*P2P$(\Pi_i^s, U_j) =$ $\mathcal{P}$.P2P(Π_j^t, U_i) *holds for any pair of partnered instances Π_i^s and Π_j^t.*

2.2 Adversarial Model and Security Goals

Security model for GKE+P protocols must address the following two challenges that are new compared to the classical GKE setting: The first challenge is to model the secrecy of a session group key k_i^s by taking into account possible leakage of any p2p key that can be computed in that session (including all $k_{i,j}^s$). Since for the secrecy of the session group key the adversary is treated as an external entity and not as a legitimate participant of that session our model should provide the adversary with the ability to schedule the on-demand computation of p2p keys and to reveal them. The second, main challenge is to model the secrecy of a session p2p key $k_{i,j}^s$ by taking into account the leakage of the group key and also the leakage of other p2p keys computed in that session (with the obvious exclusion of $k_{j,i}^t$ when Π_i^s and Π_j^t are partnered). Note that the secrecy of p2p keys does not require the adversary to be an external entity (unlike the secrecy of the group key). Hence, we have to face possible collusion attacks aiming to break the secrecy of $k_{i,j}^s$ and allow for the active participation of the adversary in the attacked session.

ADVERSARIAL MODEL. The adversary $\mathcal{A}$, modeled as a PPT machine, can schedule the protocol execution and mount own attacks via the following queries:

- *Execute*$(U_1, \ldots, U_n)$: This query executes the protocol between new instances of $U_1, \ldots, U_n \in \mathcal{U}$ and provides $\mathcal{A}$ with the execution transcript.
- *Send*(Π_i^s, m) : With this query $\mathcal{A}$ can deliver a message m to Π_i^s whereby U denotes the identity of its sender. $\mathcal{A}$ is then given the protocol message generated by Π_i^s in response to m (the output may also be empty if m is unexpected or if Π_i^s

accepts). A special invocation query of the form *Send*$(U_i, ('start', U_1, \dots, U_n))$ creates a new instance Π_i^s with $\mathtt{pid}_i^s := \{U_1, \dots, U_n\}$ and provides $\mathcal{A}$ with the first protocol message.

- *Peer*(Π_i^s, U_j): This query allows $\mathcal{A}$ to schedule the on-demand computation of p2p keys. In response, Π_i^s computes $k_{i,j}^s$; the query is processed only if Π_i^s has accepted and $U_j \in \mathtt{pid}_i^s$, and it can be asked only once per input (Π_i^s, U_j).
- *Reveal*(Π_i^s): This query models the leakage of group keys and provides $\mathcal{A}$ with k_i^s. It is answered only if Π_i^s has accepted.
- *RevealPeer*(Π_i^s, U_j): This query models the leakage of p2p keys and provides $\mathcal{A}$ with $k_{i,j}^s$; the query is answered only if *Peer*(Π_i^s, U_j) has already been asked and processed.
- *Corrupt*(U_i): This query provides $\mathcal{A}$ with LL_i. Note that in this case $\mathcal{A}$ does not gain control over the user's behavior, but might be able to communicate on behalf of the user.
- *Test*(Π_i^s): This query models indistinguishability of session group keys. Depending on a given (privately flipped) bit b $\mathcal{A}$ is given, if $b = 0$ a random session group key, and if $b = 1$ the real k_i^s. This query can be asked only once and is answered only if Π_i^s has accepted.
- *TestPeer*(Π_i^s, U_j): This query models indistinguishability of session p2p keys. Depending on a given (privately flipped) bit b $\mathcal{A}$ is given, if $b = 0$ a random session p2p key, and if $b = 1$ the real $k_{i,j}^s$. It is answered only if *Peer*(Π_i^s, U_j) has been previously asked and processed.

TERMINOLOGY. We say that U is *honest* if no *Corrupt*(U) has been asked by $\mathcal{A}$; otherwise, U is *corrupted* (or *malicious*). This also refers to the instances of U.

TWO NOTIONS OF FRESHNESS. The classical notion of freshness imposes several conditions in order to prevent any trivial break of the (A)KE-security. Obviously, we need two definitions of freshness to capture such conditions for the both key types.

First, we define the notion of *instance freshness* which will be used in the definition of (A)KE-security of group keys. Our definition is essentially the one given in [29].

Definition 2 (Instance Freshness). *An instance Π_i^s is* fresh *if Π_i^s has accepted and none of the following is true, whereby Π_j^t denotes an instance partnered with Π_i^s: (1)* Reveal(Π_i^s) *or* Reveal(Π_j^t) *has been asked, or (2)* Corrupt(U') *for some $U' \in \mathtt{pid}_i^s$ was asked before any* Send$(\Pi_i^s, \cdot)$.

Note that in the context of GKE+P the above definition restricts $\mathcal{A}$ from active participation on behalf of any user during the attacked session, but implicitly allows for the leakage of (all) p2p keys.

Additionally, we define the new notion of *instance-user freshness* which will be used to specify the (A)KE-security of p2p keys.

Definition 3 (Instance-User Freshness). *An instance-user pair (Π_i^s, U_j) is* fresh *if Π_i^s has accepted and none of the following is true, whereby Π_j^t denotes an instance partnered with Π_i^s: (1)* RevealPeer(Π_i^s, U_j) *or* RevealPeer(Π_j^t, U_i) *has been asked, or (2)* Corrupt(U_i) *or* Corrupt(U_j) *was asked before any* Send$(\Pi_i^s, \cdot)$ *or* Send$(\Pi_j^s, \cdot)$.

Here $\mathcal{A}$ is explicitly allowed to actively participate in the attacked session on behalf of any user except for U_i and U_j. Also $\mathcal{A}$ may learn the group key k_i and all p2p keys except for $k_{i,j}$. This models possible collusion of participants during the execution of the protocol aiming to break the secrecy of the p2p key $k^s_{i,j}$.

(A)KE-SECURITY OF GROUP AND P2P KEYS. For the (A)KE-security of group keys we follow the definition from [29]. Note that in case of KE-security $\mathcal{A}$ is restricted to pure eavesdropping attacks via the *Execute* query without being able to access the *Send* queries.

Definition 4 ((A)KE-Security of Group Keys). *Let $\mathcal{P}$ be a correct GKE+P protocol and b a uniformly chosen bit. By $\mathsf{Game}^{\mathsf{(a)ke\text{-}g},b}_{\mathcal{A},\mathcal{P}}(\kappa)$ we define the following adversarial game, which involves a PPT adversary $\mathcal{A}$ that is given access to all queries (except for* Send *when dealing with KE-security):*

- *$\mathcal{A}$ interacts via queries;*
- *at some point $\mathcal{A}$ asks a* Test(Π^s_i) *query for some instance Π^s_i which is (and remains) fresh;*
- *$\mathcal{A}$ continues interacting via queries;*
- *when $\mathcal{A}$ terminates, it outputs a bit, which is set as the output of the game.*

$$\text{We define:} \qquad \mathsf{Adv}^{\mathsf{(a)ke\text{-}g}}_{\mathcal{A},\mathcal{P}}(\kappa) := \left| 2\Pr[\mathsf{Game}^{\mathsf{(a)ke\text{-}g},b}_{\mathcal{A},\mathcal{P}}(\kappa) = b] - 1 \right|$$

and denote with $\mathsf{Adv}^{\mathsf{(a)ke\text{-}g}}_{\mathcal{P}}(\kappa)$ the maximum advantage over all PPT adversaries $\mathcal{A}$. We say that $\mathcal{P}$ provides (A)KE-security of group keys *if this advantage is negligible.*

Finally, we define (A)KE-security of p2p keys where we must consider possible collusion attacks. For this it is essential to allow $\mathcal{A}$ access to *Send* queries, even in the case of KE-security. The difficulty is that given general access to *Send* queries $\mathcal{A}$ can trivially impersonate any protocol participant. Hence, when dealing with KE-security of p2p keys we must further restrict $\mathcal{A}$ to truly forward all messages sent by honest users. According to our definition of instance-user freshness of (Π^s_i, U_j) this restriction will imply an unbiased communication between the instances of U_i and U_j.

Definition 5 ((A)KE-security of P2P Keys). *Let $\mathcal{P}$ be a correct GKE+P protocol and b a uniformly chosen bit. By $\mathsf{Game}^{\mathsf{(a)ke\text{-}p},b}_{\mathcal{A},\mathcal{P}}(\kappa)$ we define the following adversarial game, which involves a PPT adversary $\mathcal{A}$ that is given access to all queries (with the restriction to truly forward all messages of honest users in case of KE-security):*

- *$\mathcal{A}$ interacts via queries;*
- *at some point $\mathcal{A}$ asks a* TestPeer(Π^s_i, U_j) *query for some instance-user pair (Π^s_i, U_j) which is (and remains) fresh;*
- *$\mathcal{A}$ continues interacting via queries;*
- *when $\mathcal{A}$ terminates, it outputs a bit, which is set as the output of the game.*

$$\text{We define:} \qquad \mathsf{Adv}^{\mathsf{(a)ke\text{-}p}}_{\mathcal{A},\mathcal{P}}(\kappa) := \left| 2\Pr[\mathsf{Game}^{\mathsf{(a)ke\text{-}p},b}_{\mathcal{A},\mathcal{P}}(\kappa) = b] - 1 \right|$$

and denote with $\mathsf{Adv}^{\mathsf{(a)ke\text{-}p}}_{\mathcal{P}}(\kappa)$ the maximum advantage over all PPT adversaries $\mathcal{A}$. We say that $\mathcal{P}$ provides (A)KE-security of p2p keys *if this advantage is negligible.*

3 General Notations and Preliminaries

Throughout the paper, unless otherwise specified, by $\mathbb{G} := \langle g \rangle$ we denote a cyclic subgroup in $\mathbb{Z}_P^*$ of prime order $Q|P-1$ generated by g, where P is also prime. By $\mathtt{H_g}, \mathtt{H_p} : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ we denote two cryptographic hash functions, which will be used in our constructions for the purpose of derivation of group and p2p keys, respectively. Additionally, we recall the following three well-known cryptographic assumptions:

Definition 6 (Hardness Assumptions). *Let $\mathbb{G} := \langle g \rangle$ as above and $a, b, c \in_R \mathbb{Z}_Q$. We say that:*
The Discrete Logarithm (DL) problem is hard in $\mathbb{G}$ *if the following success probability is negligible:*

$$\mathsf{Succ}_{\mathbb{G}}^{\mathtt{DL}}(\kappa) := \max_{\mathcal{A}'} \left(\Pr_a \left[\mathcal{A}'(g, g^a) = a \right] \right);$$

The Decisional Diffie-Hellman (DDH) problem is hard in $\mathbb{G} = \langle g \rangle$ *if the following advantage is negligible:*

$$\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa) := \max_{\mathcal{A}'} \left| \Pr_{a,b} \left[\mathcal{A}'(g, g^a, g^b, g^{ab}) = 1 \right] - \Pr_{a,b,c} \left[\mathcal{A}'(g, g^a, g^b, g^c) = 1 \right] \right|;$$

The Square-Exponent Decisional Diffie-Hellman (SEDDH) problem is hard in $\mathbb{G}$[2] *if the following advantage is negligible:*

$$\mathsf{Adv}_{\mathbb{G}}^{\mathtt{SEDDH}}(\kappa) := \max_{\mathcal{A}'} \left| \Pr_a \left[\mathcal{A}'(g, g^a, g^{a^2}) = 1 \right] - \Pr_{a,b} \left[\mathcal{A}'(g, g^a, g^b) = 1 \right] \right|.$$

Note that $\mathsf{Succ}_{\mathbb{G}}^{\mathtt{DL}}(\kappa)$*,* $\mathsf{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\kappa)$*, and* $\mathsf{Adv}_{\mathbb{G}}^{\mathtt{SEDDH}}(\kappa)$ *are computed over all PPT adversaries $\mathcal{A}'$ running within time κ.*

4 Optimized PDHKE-MRE

Here we introduce our first GKE+P protocol, called PDHKE-MRE. The optimization concerns the utilization of each $x_i \in \mathbb{Z}_Q$ as a private decryption key for the multi-recipient ElGamal encryption [4, 33] *and* as a secret exponent for the computation of p2p keys via PDHKE. Note that PDHKE-MRE can be generalized by applying other multi-recipient public key encryption schemes [4]. However, in this case our optimization may no longer hold.

4.1 Parallel Diffie-Hellman Key Exchange (PDHKE)

Assuming that users interact over the authenticated channels we define PDHKE as follows (we describe all our protocols from the perspective of one session using the identities of users and not their instances):

Round 1. Each U_i chooses a random $x_i \in_R \mathbb{Z}_Q$ and broadcasts $y_i := g^{x_i}$.

[2] Wolf [41] showed that SEDDH is reducible to DDH and that the converse does not hold.

P2P key computation. Each U_i for a given identity U_j computes $k'_{i,j} := g^{x_i x_j}$ and derives $k_{i,j} := \mathtt{H_p}(k'_{i,j}, U_i|y_i, U_j|y_j)$. W.l.o.g. we assume that $i < j$ and that if U_j computes own p2p key for U_i it uses the same order for the inputs of $\mathtt{H_p}$ as U_i does.

A special attention in PDHKE should be paid to the key derivation step based on $\mathtt{H_p}$. Note that in the random oracle model this construction ensures the independence of different p2p keys (possibly computed by the same U_i for different U_j). The reason is that if U_i is honest then the hash input remains unique for each derived p2p key (due to the uniqueness of $U_i|y_i$ across different sessions and the uniqueness of each U_j within the same session). The uniqueness of hash inputs is of importance. Assume, that $k_{i,j}$ would be derived as $\mathtt{H_p}(k'_{i,j})$. In this case $\mathcal{A}$ may impose dependency between $k'_{i,j}$ and $k'_{i,a}$ for some user U_a that it may control, e.g. by using $y_a = y_j$. With this simple attack $\mathcal{A}$ cannot compute $k'_{i,a}$ due to the lack of $x_a = x_j$ but it can easily distinguish $k_{i,j}$ by obtaining $k_{i,a}$ (which would then be equal to $k_{i,j}$) via an appropriate *RevealPeer* query to an instance of honest U_i.

4.2 Multi-Recipient ElGamal Encryption (MRE)

In the classical ElGamal encryption [23] a message $m \in \mathbb{G}$ is encrypted under the recipient's public key $y = g^x$ through the computation of the ciphertext $(g^r, y^r m)$ using some random $r \in_R \mathbb{Z}_Q$. A multi-recipient ElGamal encryption (MRE) [33, 4] re-uses the random exponent r for the construction of ciphertexts of several messages $m_1, \ldots, m_n$ under several public keys $y_1 = g^{x_1}, \ldots, y_n = g^{x_n}$, i.e., by computation of $(g^r, y_1^r m_1, \ldots, y_n^r m_n)$. However, in PDHKE-MRE we will be encrypting the same message $m = m_1 = \ldots = m_n$. For this case [33] defines a computation-efficient MRE version where the ciphertext has the form $(mg^r, y_1^r, \ldots, y_n^r)$. Obviously, this technique results in shorter ciphertexts should a single protocol message contain ciphertexts for multiple recipients. Informally, the IND-CPA security of MRE means that any encrypted plaintext remains indistinguishable, even if the adversary is in possession of the secret keys $\{x_j\}_{j \neq i}$. This has been proven in [33] (and also in [4] under a stronger setting) based on the DDH assumption.

4.3 Description of PDHKE-MRE

Our optimization in PDHKE-MRE is based on the idea to re-use the same exponent x_i for both — derivation of p2p keys from $k'_{i,j} = g^{x_i x_j}$ and decryption of $\{\bar{x}_j\}_j$. The protocol PDHKE-MRE.GKE amongst a set of n users $U_1, \ldots, U_n$ proceeds in two rounds:

Round 1. Each U_i chooses a random $x_i \in_R \mathbb{Z}_Q$ and broadcasts $y_i := g^{x_i}$.
Round 2. Each U_i chooses random $\bar{x}_i \in_R \mathbb{G}$, $r_i \in_R \mathbb{Z}_Q$, computes $z_i := \bar{x}_i g^{r_i}$ and $\{z_{i,j} := y_j^{r_i}\}_j$ and broadcasts $(z_i, \{z_{i,j}\}_j)$.

Group key computation. Each U_i decrypts $\left\{\bar{x}_j := \frac{z_j}{z_{j,i}^{(1/x_i)}}\right\}_j$ and accepts with $k_i := \mathtt{H_g}(\bar{x}_1, \ldots, \bar{x}_n,\ \mathtt{sid}_i)$ where $\mathtt{sid}_i := (U_1|y_1, \ldots, U_n|y_n)$.

The algorithm PDHKE-MRE.P2P when executed by some user U_i for a peer U_j computes $k'_{i,j} := g^{x_i x_j}$ and outputs $k_{i,j} := \mathtt{H_p}(k'_{i,j}, U_i|y_i, U_j|y_j)$ whereby the inputs $U_i|y_i$

and $U_j|y_j$ are taken from $\mathtt{sid}_i$. W.l.o.g. we assume that $i < j$ and that U_j will use the same order for the inputs to $\mathtt{H_p}$ in the computation of $k_{j,i}$.

4.4 Security Analysis of PDHKE-MRE

Although the stand-alone security of MRE can be proven under the DDH assumption, its optimized merge with PDHKE requires the additional use of the SEDDH assumption for the proof of KE-security of group keys as motivated in the following.

The natural way to prove the IND-CPA security of MRE under the DDH assumption would be to simulate $y_j = g^{a\alpha_j}$, $z_i = \bar{x}_i g^{b\beta_i}$, and each $z_{i,j} = g^{ab\alpha_j\beta_i}$, where g^a and g^b belong to the DDH tuple and $\alpha_j, \beta_i \in_R \mathbb{Z}_Q$ (observe that the DDH problem is self-reducible). However, in PDHKE-MRE this simulation would also mean that $y_i = g^{a\alpha_i}$ for some $\alpha_i \in_R \mathbb{Z}_Q$ and possibly imply $g^{x_i x_j} = g^{a^2\alpha_i\alpha_j}$ upon the simulation of p2p keys, which in turn involves g^{a^2} from the SEDDH tuple.

Theorem 1. *If both problems DDH and SEDDH are hard in* $\mathbb{G}$ *then* PDHKE-MRE *provides KE-security of group keys and*

$$\mathsf{Adv}^{\mathsf{ke\text{-}g}}_{\text{PDHKE-MRE}}(\kappa) \le \frac{2(N(\mathsf{q}_{\mathtt{Ex}} + \mathsf{q}_{\mathsf{Se}})^2 + \mathsf{q}_{\mathtt{H_g}})}{Q} + \frac{(\mathsf{q}_{\mathtt{H_g}} + \mathsf{q}_{\mathtt{H_p}})^2}{2^{\kappa-1}} + 2N\mathsf{Adv}^{\mathtt{SEDDH}}_{\mathbb{G}}(\kappa) + 2N(N-1)\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa)$$

with at most $(\mathsf{q}_{\mathtt{Ex}} + \mathsf{q}_{\mathsf{Se}})$ *sessions being invoked via Execute and Send queries and at most* $\mathsf{q}_{\mathtt{H_g}}$ *and* $\mathsf{q}_{\mathtt{H_p}}$ *random oracle queries being asked.*

Since secret contributions $\bar{x}_i$ used in the computation of the group key are independent from the secret exponents x_i we can prove that PDHKE-MRE provides KE-security of p2p keys based on the DDH assumption.

Theorem 2. *If the DDH problem is hard in* $\mathbb{G}$ *then* PDHKE-MRE *provides KE-security of p2p keys and*

$$\mathsf{Adv}^{\mathsf{ke\text{-}p}}_{\text{PDHKE-MRE}}(\kappa) \le \frac{N(2(\mathsf{q}_{\mathtt{Ex}} + \mathsf{q}_{\mathsf{Se}})^2 + \mathsf{q}_{\mathsf{Se}}\mathsf{q}_{\mathtt{H_p}})}{Q} + \frac{(\mathsf{q}_{\mathtt{H_g}} + \mathsf{q}_{\mathtt{H_p}})^2}{2^{\kappa-1}} + N\mathsf{q}_{\mathsf{Se}}\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa)$$

with at most $(\mathsf{q}_{\mathtt{Ex}} + \mathsf{q}_{\mathsf{Se}})$ *sessions being invoked via Execute and Send queries and at most* $\mathsf{q}_{\mathtt{H_g}}$ *and* $\mathsf{q}_{\mathtt{H_p}}$ *random oracle queries being asked.*

4.5 On Security of PDHKE as a Stand-Alone Protocol

The result of Theorem 2 allows us to derive the following corollary, which is of independent interest since it addresses security of PDHKE as a stand-alone protocol.

Corollary 1. *If the DDH problem is hard in* $\mathbb{G}$ *then* PDHKE *as defined in Section 4.1 guarantees the KE-security of p2p keys in the random oracle model in the sense of Definition 5.*[3]

[3] Observe that our security model can be used to deal with PDHKE as a stand-alone protocol assuming that in the execution of PDHKE instances accept with empty group keys. In this case all parts of the model that explicitly deal with the computation and security of group keys become irrelevant.

4.6 Performance Limitations of PDHKE-MRE

The drawback of PDHKE-MRE despite of our optimizations is the quadratic *communication complexity*, i.e. the total number of bits communicated throughout the protocol and usually measured in the size of group (or public key) elements [29]. This complexity is due to the rather naïve secure transport of each $\bar{x}_i$ for the computation of the group key. Note that the linear communication complexity of PDHKE used to compute p2p keys is already optimal since each user has to broadcast at least one message in order to contribute to the on-demand computation of its p2p keys.

Therefore, we will try to replace the computation of the group key via MRE with an alternative process, while preserving the computation of p2p keys based on PDHKE. Since PDHKE derives p2p keys from Diffie-Hellman secrets it appears promising to search for alternative candidates amongst the family of GroupDH protocols, i.e. GKE protocols that extend the original Diffie-Hellman method.

5 GKE+P Protocols from Group Diffie-Hellman Protocols

We start by describing a generic solution that would convert any secure GroupDH protocol into a secure GKE+P protocol. Then, we address possible optimization issues.

5.1 GKE+P Compiler Based on PDHKE

Let us first capture the similarities between different GroupDH protocols by providing a generalized definition of what a GroupDH protocol should mean (we define from the perspective of one session).

Definition 7 (GroupDH Protocols). *A* GroupDH protocol *is a GKE protocol amongst n users $U_1, \ldots, U_n$ such that during its execution each user U_i chooses own exponent $x_i \in_R \mathbb{Z}_Q$ and at the end computes a group element $k_i' \in \mathbb{G}$ which can be expressed as the output of $f(g, x_1, \ldots, x_n)$ for some function $f : \mathbb{G} \times \mathbb{Z}_Q^n \rightarrow \mathbb{G}$ which is specific to the protocol.*

We say that a GroupDH protocol is KE-secure *if it achieves KE-security of group keys in the sense of Definition 4 whereby considering k_i' instead of k_i and thus requiring its indistinguishability from some random element in $\mathbb{G}$ instead of some random string in $\{0,1\}^\kappa$.*[4]

The above definition of KE-secure GroupDH protocols already captures many protocols, including those from [16, 20, 24, 31, 32, 37, 39, 40].

The actual generic solution (GKE+P compiler) for obtaining a GKE+P protocol from such GroupDH protocols is to combine them with PDHKE, while ensuring independence between the exponents used in both protocols. More precisely, GKE+P compiler requires each user U_i to choose a random exponent $\bar{x}_i \in_R \mathbb{Z}_Q$ and broadcast $\bar{y}_i := g^{\bar{x}_i}$ prior to the execution of the given GroupDH protocol. If the GroupDH protocol requires each user to broadcast a message in the first round, e.g. [16, 31, 32, 39], then the compiler can also append $\bar{y}_i$ to this first message, without increasing the

[4] Note that Definition 4 can be easily adapted by the appropriate modification of the *Test* query.

number of rounds. After the GroupDH protocol is executed each U_i holds the secret group element k'_i. The GKE+P compiler computes $\mathtt{sid}_i := (U_1|\bar{y}_1, \ldots, U_n|\bar{y}_n)$ and derives the group key $k_i := \mathtt{H}_\mathtt{g}(k'_i, \mathtt{sid}_i)$. On-demand, the compiler computes any $k_{i,j} := \mathtt{H}_\mathtt{p}(\bar{y}_j^{\bar{x}_i}, U_i|\bar{y}_i, U_j|\bar{y}_j)$.

The key derivation is essentially the same as in PDHKE-MRE. The only difference is that $\mathtt{sid}_i$ is constructed from $\bar{y}_i$ instead of $y_i = g^{x_i}$ for the exponent x_i which is implicit to the original GroupDH protocol. The reason is that y_i may not be available to all users at the end of the protocol. For example, in [24,40] only two users U_1 and U_2 compute such y_1 and y_2, whereas in [37,20] each U_i computes y_i but sends it only to some designated subset. Of course, for the latter case it is possible to add a modification to the original protocol by requiring users to broadcast y_i; however, this contradicts to the idea of a compiler, which takes some protocol as a "black-box".

The KE-security of group keys output by our compiler follows from the KE-security of the group elements k' and can be proven similarly to Theorem 1. Note that the replacement of y_i with $\bar{y}_i$ in the computation of $\mathtt{sid}_i$ has no impact since also $\bar{y}_i$ is uniformly distributed in $\mathbb{G}$ for any honest U_i. Since the exponents x_i and $\bar{x}_i$ are independent and values $\bar{y}_i$ and $\bar{y}_j$ exchanged between any two honest users U_i and U_j are not modified during the transmission (as required by our model) the KE-security of computed p2p keys would follow directly from Corollary 1. We omit the detailed analysis of the GKE+P compiler, which seems fairly natural.

Instead, we focus on the next challenge and investigate whether GroupDH protocols can be merged with PDHKE in order to obtain possibly more efficient GKE+P protocols than those given by our generic compiler. Can we find suitable GroupDH protocols where the implicitly used exponents $x_1, \ldots, x_n$ can be safely *re-used* for the computation of p2p keys? Intuitively, this question should be answered separately for each GroupDH protocol. Due to space limitations, we restrict our analysis to two well-known protocols from [16] and [31] that implicitly require each U_i to broadcast $y_i := g^{x_i}$ and so seem suitable at first sight for the merge with PDHKE.

5.2 PDHKE-BD Is Insecure

The Burmester-Desmedt (BD) protocol from [16] is one of the best known unauthenticated GroupDH protocols. It has been formally proven KE-secure under the DDH assumption in [29]. Its technique has influenced many GKE protocols, including [30,2]. The BD protocol arranges participants $U_1, \ldots, U_n$ into a cycle, and requires two communication rounds:

Round 1. Each U_i broadcasts $y_i := g^{x_i}$ for some random $x_i \in_R \mathbb{Z}_Q$.
Round 2. Each U_i broadcasts $z_i := (y_{i+1}/y_{i-1})^{x_i}$ (the indices i form a cycle, i.e. $0 = n$ and $n + 1 = 1$).

This allows each U_i to compute the secret group element

$$k'_i := (y_{i-1})^{nx_i} \cdot z_i^{n-1} \cdot z_{i+1}^{n-2} \cdots z_{i+n-2} = g^{x_1x_2+x_2x_3+\ldots+x_nx_1}.$$

At first sight, BD suits for the merge with PDHKE, i.e. we would have then $k'_i := \mathtt{H}_\mathtt{g}(k'_i, U_1|y_1, \ldots, U_n|y_n)$ and any $k_{i,j} := \mathtt{H}_\mathtt{p}(y_j^{x_i}, U_i|y_i, U_j|y_j)$. Unfortunately, this merge is insecure. We analyze two distinct cases based on the indices of U_i and U_j.

CASE U_i AND U_{i+1}. The attack in this case is trivial since the knowledge of k' and the secret exponents of all other colluding users allows to compute $g^{x_ix_{i+1}}$. This would break the secrecy of the p2p key $k_{i,i+1}$ when derived using $g^{x_ix_{i+1}}$ for any group size $n \geq 3$. Also observe that each U_i sends $z_i = g^{x_{i+1}x_i - x_ix_{i-1}}$; thus every U_{i-1} can individually extract $g^{x_{i+1}x_i}$ and every U_{i+1} is able to compute $g^{x_ix_{i-1}}$, even without colluding with other users.

CASE U_i AND U_j. In this case we consider $k_{i,j}$ (w.l.o.g. we assume that $i < j$) computed for a pair of users that do not have neighbor positions within the cycle, i.e. $j \neq i + 1$. We demonstrate that also this key remains insecure if derived using $g^{x_ix_j}$. Our attack, which is not as trivial as in the previous case, works because users may collude and misbehave while attacking the secrecy of p2p keys. In particular, we assume that U_{i-2}, U_{i-1}, and U_{i+1} collude and their goal is to obtain $g^{x_ix_j}$ upon the successful execution of the protocol from the perspective of honest U_i and U_j. Due to the collusion of three users the attack works for any group size $n > 4$. The core of the attack is to let U_{i-1} broadcast $y_{i-1} := y_j$, which is possible since the communication is asynchronous and $\mathcal{A}$ can wait for the protocol message of U_j containing y_j; observe that x_j is chosen by U_j and remains unknown to the colluding users. Other malicious users U_{i-2} and U_{i+1} choose their exponents x_{i-2} and x_{i+1} truly at random. As a consequence, in the second round honest U_i broadcasts $z_i = g^{x_{i+1}x_i - x_ix_{i-1}} = g^{x_{i+1}x_i - x_ix_j}$. Then, malicious U_{i+1} can extract $g^{x_ix_j} := y_i^{x_{i+1}}/z_i$. Finally, U_{i-1} without knowing the corresponding exponents x_j and x_i has to broadcast a value of the form $z_{i-1} = g^{x_ix_{i-1} - x_{i-1}x_{i-2}} = g^{x_ix_j - x_jx_{i-2}}$ which can be easily done with the assistance of U_{i+1} that provides $g^{x_ix_j}$ and of U_{i-1} that provides $g^{x_jx_{i-2}} = y_j^{x_{i-2}}$. Thus, through their cooperation malicious users U_{i-2}, U_{i-1}, and U_{i+1} can extract $g^{x_ix_j}$ for any U_j. The above attacks works similarly even if U_{i-1} re-randomizes y_j, i.e. broadcasts $y_{i-1} = y_j^r$ for some $r \in_R \mathbb{Z}_Q$.

This shows that BD cannot be merged with PDHKE in a secure way. Nevertheless, it can be compiled to a KE-secure GKE+P protocol as discussed in Section 5.1.

5.3 PDHKE-KPT Is Secure

Here we focus on the GKE protocols proposed by Kim, Perrig, and Tsudik [31, 32], which in turn extend the less efficient construction by Steer et al. [39]. These protocols belong to a family of the so-called Tree Diffie-Hellman protocols (see also [22,14]). We analyze whether the protocol from [31], denoted here as KPT, which is more efficient in communication than [32], can be securely merged with PDHKE.

The KPT protocol requires a special group $\mathbb{G} = \langle g \rangle$ of prime order Q, which is a group of quadratic residues modulo a safe prime $P = 2Q+1$ with the group law defined as $ab := f(ab \mod P)$ for any $a, b \in \mathbb{G}$ where $f : \mathbb{Z}_P \mapsto \mathbb{Z}_Q$ is such that if $z \leq Q$ then $f(z) := z$, otherwise if $Q < z < P$ then $f(z) := P - z$ (see [31, 32, 14] for more information about $\mathbb{G}$ which equals to $\mathbb{Z}_Q$ as sets). In KPT each U_i derives the secret group element k'_i within two communication rounds (it is assumed that the sequence $U_1, \ldots, U_n$ is ordered):

Round 1. Each U_i broadcasts $y_i := g^{x_i}$ for some random $x_i \in_R \mathbb{Z}_Q$.

Round 2. U_1 computes and broadcasts $(g^{z_2}, \dots, g^{z_{n-1}})$ whereby $z_2 := y_2^{x_1}$ and each $z_i := y_i^{z_{i-1}}$ for all $i = 3, \dots, n-1$.

This allows each U_i to compute the common secret $k'_i := z_n$ as follows.

- U_1 computes $k'_1 := y_n^{z_{n-1}}$
- each U_i, $2 \le i \le n-1$ recomputes the subsequence $z_i, \dots, z_{n-1}$ and computes $k'_i := y_n^{z_{n-1}}$; note that U_2 starts with $z_2 := y_1^{x_2}$, whereas U_i, $3 \le i \le n-1$, starts with $z_i := (g^{z_{i-1}})^{x_i}$ using $g^{z_{i-1}}$ received from U_1.
- U_n computes $k'_i := (g^{z_{n-1}})^{x_n}$ using $g^{z_{n-1}}$ received from U_1.

Note that each k'_i has an interesting algebraic structure

$$k'_i = g^{x_n g^{x_{n-1} g^{\cdots g^{x_3 g^{x_2 x_1}}}}}.$$

In the following we investigate the possibility of merging KPT with PDHKE, thus using exponents x_i to compute the group key $k_i := \mathtt{H_g}(k'_i, U_1|y_1, \dots, U_n|y_n)$ and any p2p key $k_{i,j} := \mathtt{H_g}(k'_{i,j}, U_i|y_i, \dots, U_j|y_j)$ with $k'_{i,j} = g^{x_i x_j}$. Our analysis shows that indeed this construction, which we denote PDHKE-KPT, gives us a KE-secure GKE+P protocol.

Let us first provide some intuition. Note that the only value of the form $g^{x_i x_j}$ which appears in the computations of KPT is $g^{x_1 x_2}$ (given by z_2). Nevertheless, it will be computed only by U_1 and U_2, which is fine since the p2p key should be known only to these users. Further we observe that the broadcast message of U_1 contains $g^{z_2} = g^{g^{x_1 x_2}}$ and so hides $g^{x_1 x_2}$ in the exponent (under the hardness of the DL problem). By computing $k_{1,2} := \mathtt{H_p}(g^{x_1 x_2}, U_1|y_1, U_2|y_2)$ we are able to provide independence between $k_{1,2}$ and g^{z_2} while working in the random oracle model since the corresponding *RevealPeer* query would reveal only $k_{1,2}$ and not $g^{x_1 x_2}$.

We start with the KE-security of group keys. The original KPT protocol has been proven KE-secure in [31] (see also [14]) under the classical DDH assumption. Briefly, the proof considers several hybrid games. In the l-th game, $2 \le l \le n$, the simulator embeds a re-randomized DDH tuple (g, g^a, g^b, g^{ab}) to simulate $g^{z_{l-1}} = g^{a\alpha_{l-1}}$, $y_i = g^{b\beta_l}$, and $z_l = g^{ab\alpha_{l-1}\beta_l}$, such that in the final game the value $z_n = k'_i$ is uniformly distributed and independent. In general we can apply a similar simulation technique, however, we should additionally take care of the special dependency $z_2 = k'_{1,2}$. The trick is first to obtain a uniform distribution of $z_2 = k'_{1,2}$ (in $\mathbb{G}$) and its independence from y_1 and y_2 using the above technique and then to compute $k_{1,2}$ completely independent from $k'_{1,2}$, in which case a reduction to the DL problem becomes possible.

Theorem 3. *If both problems DDH and DL are hard in* $\mathbb{G}$ *then* PDHKE-KPT *provides KE-security of group keys and*

$$\mathsf{Adv}^{\mathsf{ke\text{-}g}}_{\text{PDHKE-KPT}}(\kappa) \le \frac{2(N(\mathsf{q_{Ex}} + \mathsf{q_{Se}})^2 + \mathsf{q}_{\mathtt{H_g}})}{Q} + \frac{(\mathsf{q}_{\mathtt{H_g}} + \mathsf{q}_{\mathtt{H_p}})^2}{2^{\kappa-1}}$$
$$+2(N-1)\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) + 2\mathsf{q}_{\mathtt{H_p}}\mathsf{Succ}^{\mathtt{DL}}_{\mathbb{G}}(\kappa)$$

with at most $(\mathsf{q_{Ex}} + \mathsf{q_{Se}})$ *sessions being invoked via Execute and Send queries and at most* $\mathsf{q}_{\mathtt{H_g}}$ *and* $\mathsf{q}_{\mathtt{H_p}}$ *random oracle queries asked.*

Finally, we prove that on-demand p2p keys computed in PDHKE-KPT are also KE-secure. In general we can follow the proof of Theorem 2 based on the DDH assumption, however, we have also to take care of the special case $(i, j) = (1, 2)$. Observe that if $k_{1,2}$ becomes a subject of the attack then U_1 and U_2 must be honest, in which case we can still apply the above trick.

Theorem 4. *If both problems DDH and DL are hard in* $\mathbb{G}$ *then* PDHKE-KPT *provides KE-security of p2p keys and*

$$\mathsf{Adv}^{\mathsf{ke\text{-}p}}_{\mathrm{PDHKE\text{-}KPT}}(\kappa) \leq \frac{N(2(\mathsf{q}_{\mathtt{Ex}} + \mathsf{q}_{\mathtt{Se}})^2 + \mathsf{q}_{\mathtt{Se}}\mathsf{q}_{\mathrm{H_p}})}{Q} + \frac{(\mathsf{q}_{\mathrm{H_g}} + \mathsf{q}_{\mathrm{H_p}})^2}{2^{\kappa-1}} + N\mathsf{q}_{\mathtt{Se}}\left(\mathsf{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\kappa) + \mathsf{q}_{\mathrm{H_p}}\mathsf{Succ}^{\mathtt{DL}}_{\mathbb{G}}(\kappa)\right)$$

with at most $(\mathsf{q}_{\mathtt{Ex}} + \mathsf{q}_{\mathtt{Se}})$ *sessions being invoked via Execute and Send queries and at most* $\mathsf{q}_{\mathrm{H_g}}$ *and* $\mathsf{q}_{\mathrm{H_p}}$ *random oracle queries asked.*

6 Performance Comparison and Discussion

In Table 1 we present a brief comparison of the complexity of the mentioned GKE+P solutions. We measure the communication costs as a total number of transmitted elements in $\mathbb{G}$, and computation costs as a number of modular exponentiations *per* U_i (in the case of BD we count only exponentiations with x_i assuming that $|x_i| \gg n$). From the latter we exclude the costs needed to compute a Diffie-Hellman secret $k'_{i,j}$ that requires constantly one exponentiation per each U_j. For the GKE+P compiler from Section 5.1 with the prefix '+' we indicate the increase to the original costs of the given GroupDH protocol when combined with PDHKE; we also mention the compiled GKE+P version of the BD protocol as a special case. Note that the PDHKE-KPT protocol has asymmetric costs, depending on the position of U_i in the ordered sequence $U_1, \ldots, U_n$; this may have benefits in groups with heterogeneous devices.

Table 1. Communication and Computation Costs of Introduced GKE+P Protocols

GKE+P Protocols	Communication (in log Q bits)	Computation (in mod. exp. per U_i)
PDHKE-MRE	$n^2 + n$	$2n$
GKE+P compiler	$+n$	$+1$
BD (as a special case)	$3n$	3
PDHKE-KPT	$2n - 2$	$n + 2 - i$ ($2n - 2$ for U_1)

From Table 1 we highlight that PDHKE-KPT has better communication complexity than the compiled version of the BD protocol, but (in general much) worse computation complexity. The same holds for the original KPT and BD protocols. Therefore, we do not claim that GroupDH protocols when merged with PDHKE in an optimized way (via exponent re-use) would result in more efficient constructions compared to other protocols obtained via our GKE+P compiler. Nevertheless, with PDHKE-KPT we could show that there exist GKE protocols that provide the property of non-interactive, on-demand computation of p2p keys almost “for free” (if one neglects the computation costs needed for the derivation of keys then the costs of PDHKE-KPT from Table 1 are identical to those of KPT).

7 Adding Authentication to GKE+P Protocols

Yet, we were assuming that described GKE+P protocols are executed over authenticated links and focused on the KE-security of their group and p2p keys. On the other hand, it is well-known that any KE-secure GKE protocol can be converted into an AKE-secure protocol (preserving its forward secrecy) using the classical and inexpensive compilation technique from [29] which assumes for each user U_i a long-lived digital signature key pair (sk_i, pk_i) such that in the preliminary protocol round users exchange their nonces r_i and then sign each l-th round message m_l concatenated with $U_1|r_1|\ldots|U_n|r_n$ prior to the transmission. The EUF-CMA security of the digital signature and the negligible collision probability for the nonces protects against impersonation and replay attacks.

The following theorem shows that this technique is also sufficient to obtain AKE-security of group and p2p keys in GKE+P protocols.

Theorem 5. *If $\mathcal{P}$ is a GKE+P protocol that provides KE-security of group/p2p keys then $\mathcal{P}$ compiled with the technique from [29] results in a GKE+P protocol $\mathcal{P}'$ that provides AKE-security of group/p2p keys.*

Proof Idea: Theorem 5 can be proven in two steps (one for group keys, another one for p2p keys) using the same strategy as in the proof of [29, Theorem 2]. Briefly, in each of the both steps the proof first eliminates signature forgeries and replay attacks and then constructs an adversary $\mathcal{A}$ against the KE-security of group/p2p keys that interacts with the user instances and also simulates the additional authentication steps while answering the queries of an adversary $\mathcal{A}'$ against the AKE-security of group/p2p keys. In case of group keys $\mathcal{A}$ will need to guess the session in which the *Test*(Π_i^s) query will be asked in order to simulate the protocol execution in that session through the authentication of the transcript, which $\mathcal{A}$ obtains initially via own *Execute* query. In case of p2p keys $\mathcal{A}$ will need to guess the session in which the *TestPeer*(Π_i^s, U_j) query will be asked *and* two corresponding identities U_i and U_j of honest users in order to add authentication to their messages, which $\mathcal{A}$ obtains by relaying the *Send* queries of $\mathcal{A}'$. We omit the details.

8 Conclusion

We discussed the enrichment of GKE protocols with the property of non-interactive, on-demand derivation of peer-to-peer keys, which allows for the establishment of a secure group channel and up to n independently secure peer-to-peer channels through a single run of the protocol. We extended the standard GKE security model capturing independence of group and p2p keys as well as possible collusion attacks against the secrecy of the latter and proposed several provably secure solutions with varying efficiency. With PDHKE-KPT we demonstrated the existence of GKE protocols that implicitly allow derivation of p2p keys without any increase of their original communication complexity. Future work may include consideration of the optional insider threats against the group keys computed in GKE+P protocols in the spirit of [28,12,13]. Another interesting direction is to investigate to what extent (x_i, g^{x_i}) often computed in GroupDH protocols can be used as key pairs in digital signatures, public-key encryption schemes, etc.

References

1. Abdalla, M., Bohli, J.-M., Vasco, M.I.G., Steinwandt, R.: (Password) Authenticated Key Establishment: From 2-Party to Group. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 499–514. Springer, Heidelberg (2007)
2. Abdalla, M., Bresson, E., Chevassut, O., Pointcheval, D.: Password-Based Group Key Exchange in a Constant Number of Rounds. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 427–442. Springer, Heidelberg (2006)
3. Abdalla, M., Catalano, D., Chevalier, C., Pointcheval, D.: Efficient Two-Party Password-Based Key Exchange Protocols in the UC Framework. In: Malkin, T.G. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 335–351. Springer, Heidelberg (2008)
4. Bellare, M., Boldyreva, A., Staddon, J.: Randomness Re-use in Multi-recipient Encryption Schemeas. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 85–99. Springer, Heidelberg (2003)
5. Bellare, M., Canetti, R., Krawczyk, H.: A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. In: ACM STOC 1998, pp. 419–428. ACM Press, New York (1998)
6. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure Against Dictionary Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
7. 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)
8. Biswas, G.P.: Diffie-Hellman Technique: Extended to Multiple Two-Party Keys and One Multi-Party Key. IET Inf. Sec. 2(1), 12–18 (2008)
9. Boyd, C., Mathuria, A.: Protocols for Authentication and Key Establishment. Springer, Heidelberg (2003)
10. Bresson, E., Chevassut, O., Pointcheval, D.: Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 321–336. Springer, Heidelberg (2002)
11. Bresson, E., Chevassut, O., Pointcheval, D., Quisquater, J.-J.: Provably Authenticated Group Diffie-Hellman Key Exchange. In: ACM CCS 2001, pp. 255–264. ACM Press, New York (2001)
12. Bresson, E., Manulis, M.: Malicious Participants in Group Key Exchange: Key Control and Contributiveness in the Shadow of Trust. In: Xiao, B., Yang, L.T., Ma, J., Muller-Schloer, C., Hua, Y. (eds.) ATC 2007. LNCS, vol. 4610, pp. 395–409. Springer, Heidelberg (2007)
13. Bresson, E., Manulis, M.: Contributory Group Key Exchange in the Presence of Malicious Participants. IET Inf. Sec. 2(3), 85–93 (2008)
14. Bresson, E., Manulis, M.: Securing Group Key Exchange against Strong Corruptions. In: ACM ASIACCS 2008, pp. 249–260. ACM Press, New York (2008)
15. Bresson, E., Manulis, M., Schwenk, J.: On Security Models and Compilers for Group Key Exchange Protocols. In: Miyaji, A., Kikuchi, H., Rannenberg, K. (eds.) IWSEC 2007. LNCS, vol. 4752, pp. 292–307. Springer, Heidelberg (2007)

16. Burmester, M., Desmedt, Y.: A Secure and Efficient Conference Key Distribution System. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 275–286. Springer, Heidelberg (1995)
17. 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)
18. Canetti, R., Krawczyk, H.: Universally Composable Notions of Key Exchange and Secure Channels. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 337–351. Springer, Heidelberg (2002)
19. Choo, K.-K.R., Boyd, C., Hitchcock, Y.: Examining Indistinguishability-Based Proof Models for Key Establishment Protocols. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 585–604. Springer, Heidelberg (2005)
20. Desmedt, Y., Lange, T.: Revisiting Pairing Based Group Key Exchange. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 53–68. Springer, Heidelberg (2008)
21. Diffie, W., Hellman, M.E.: New Directions in Cryptography. IEEE Tran. on Inf. Th. 22(6), 644–654 (1976)
22. Dutta, R., Barua, R., Sarkar, P.: Provably Secure Authenticated Tree Based Group Key Agreement. In: López, J., Qing, S., Okamoto, E. (eds.) ICICS 2004. LNCS, vol. 3269, pp. 92–104. Springer, Heidelberg (2004)
23. Gamal, T.E.: 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)
24. Ingemarsson, I., Tang, D.T., Wong, C.K.: A Conference Key Distribution System. IEEE Tran. on Inf. Th. 28(5), 714–719 (1982)
25. Jarecki, S., Kim, J., Tsudik, G.: Authentication for Paranoids: Multi-party Secret Handshakes. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 325–339. Springer, Heidelberg (2006)
26. Jarecki, S., Kim, J., Tsudik, G.: Group Secret Handshakes Or Affiliation-Hiding Authenticated Group Key Agreement. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 287–308. Springer, Heidelberg (2007)
27. Jeong, I.R., Lee, D.H.: Parallel Key Exchange. J. of Univ. Comp. Sci. 14(3), 377–396 (2008)
28. Katz, J., Shin, J.S.: Modeling Insider Attacks on Group Key-Exchange Protocols. In: ACM CCS 2005, pp. 180–189. ACM Press, New York (2005)
29. Katz, J., Yung, M.: Scalable Protocols for Authenticated Group Key Exchange. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 110–125. Springer, Heidelberg (2003)
30. Kim, H.-J., Lee, S.-M., Lee, D.H.: Constant-Round Authenticated Group Key Exchange for Dynamic Groups. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 245–259. Springer, Heidelberg (2004)
31. Kim, Y., Perrig, A., Tsudik, G.: Group Key Agreement Efficient in Communication. IEEE Tran. on Comp. 53(7), 905–921 (2004)
32. Kim, Y., Perrig, A., Tsudik, G.: Tree-Based Group Key Agreement. ACM Trans. on Inf. and Syst. Sec. 7(1), 60–96 (2004)
33. Kurosawa, K.: Multi-Recipient Public-Key Encryption with Shortened Ciphertext. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 48–63. Springer, Heidelberg (2002)
34. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger Security of Authenticated Key Exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007)
35. Manulis, M.: Security-Focused Survey on Group Key Exchange Protocols. Cryptology ePrint Archive, Report 2006/395 (2006)

36. Mayer, A., Yung, M.: Secure Protocol Transformation via "Expansion": From Two-Party to Groups. In: ACM CCS 1999, pp. 83–92. ACM Press, New York (1999)
37. Nam, J., Paik, J., Kim, U.-M., Won, D.: Constant-Round Authenticated Group Key Exchange with Logarithmic Computation Complexity. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 158–176. Springer, Heidelberg (2007)
38. Shoup, V.: On Formal Models for Secure Key Exchange (Version 4). TR RZ 3120, IBM Research (1999)
39. Steer, D.G., Strawczynski, L., Diffie, W., Wiener, M.J.: A Secure Audio Teleconference System. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 520–528. Springer, Heidelberg (1990)
40. Steiner, M., Tsudik, G., Waidner, M.: Diffie-Hellman Key Distribution Extended to Group Communication. In: ACM CCS 1996, pp. 31–37. ACM Press, New York (1996)
41. Wolf, S.: Information-Theoretically and Computationally Secure Key Agreement in Cryptography. PhD thesis, ETH Zürich (1999)

Session-state Reveal Is Stronger Than Ephemeral Key Reveal: Attacking the NAXOS Authenticated Key Exchange Protocol

Cas J.F. Cremers[*]

Department of Computer Science, ETH Zurich
8092 Zurich, Switzerland
cas.cremers@inf.ethz.ch

Abstract. In the paper "Stronger Security of Authenticated Key Exchange" [1,2], a new security model for authenticated key exchange protocols (eCK) is proposed. The new model is suggested to be at least as strong as previous models for key exchange protocols. The model includes a new notion of an Ephemeral Key Reveal adversary query, which is claimed in e. g. [2,3,4] to be at least as strong as the Session-state Reveal query. We show that Session-state Reveal is stronger than Ephemeral Key Reveal, implying that the eCK security model is incomparable to the CK model [5,6]. In particular we show that the proposed NAXOS protocol from [1,2] does not meet its security requirements if the Session-state Reveal query is allowed in the eCK model. We discuss the implications of our result for some related protocols proven correct in the eCK model, and discuss the interaction between Session-state Reveal and protocol transformations.

Keywords: Provably-secure, Authenticated Key Exchange, Session-state reveal, Ephemeral Key reveal.

1 Introduction

In the area of secure key agreement protocols many security models [7,8,1,9,5,10] and protocols have been proposed. Many of the proposed protocols have been shown to be correct in some particular security model, but have also shown to be incorrect in others. In order to determine the exact properties that are required from such protocols, a single unified security model would be desirable. However, given the recent works such as [8], it seems that a single model is still not agreed upon.

In this paper we focus on a specific aspect of security models for key agreement protocols. In particular, we focus on the ability of the adversary to learn the local state of an agent. For example, when an agent chooses a random value, or computes the hash function of a certain input, the constituents of the computation reside temporarily in the local memory of the agent. It may be possible for the

[*] This work was supported by the Hasler Foundation within the ComposeSec project.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 20–33, 2009.

adversary to learn such information, even though he cannot learn the long-term private keys of the agent. This corresponds to the situation in which the long-term private keys reside in e. g. a tamper-proof module (TPM) or cryptographic coprocessor, while the remainder of the protocol computations are done in regular (unprotected) memory. The corresponding adversary ability is captured in security models for key agreement protocols by the Session-state Reveal query.

A drawback of the Session-state Reveal query in current security models is that the query is often underspecified. For example, in the Canetti-Krawczyk (CK) model [5], Session-state Reveal is defined as giving the adversary the internal state of the Turing machine that executes the protocol. This internal state is not defined within the security model. Effectively, the definition of the internal state is postponed to the proof of a particular protocol.

In [2,1] a security model is proposed which is said to be stronger than existing AKE (Authenticated Key Exchange) security models. The model is based on the CK model, and is referred to in [1] as the Extended Canetti-Krawczyk (eCK) model. The eCK model differs in a number of aspects from the CK model, where the main difference seems to be that the adversary is allowed to reveal part of the local state of participants even during a normal protocol session. A more subtle aspect in which the eCK model differs from the CK model is that it replaces the Session-state Reveal query by a new Ephemeral Key Reveal query. In this paper we focus on this aspect.

In order to provide a definition of the local state within the security model, the eCK model (re)defines the notion of ephemeral key and introduces a corresponding Ephemeral Key Reveal query that reveals this key. The ephemeral key is defined to contain all secret session-specific information. The authors argue for the new Ephemeral Key Reveal query that "by setting the ephemeral secret key equal to all session-specific secret information, we seem to cover all definitions of Session-state Reveal queries which exist in literature" [2, p. 2]. Similar arguments can be found in [4,3,11,12]. Within the resulting eCK model, the NAXOS protocol is proposed and proven correct in [1].

Contrary to the above, it is argued in [13] that strictly speaking the eCK and CK models are incomparable. Regarding the difference between Session-state Reveal and Ephemeral Key Reveal, it is remarked that "The important point to note is that the ephemeral-key does not include session state that has been computed using the long-term secret of the party. This is not the case in the CK model where, in principle, the adversary is allowed access to all the inputs (including the randomness, but excluding the long-term secret itself) and the results of all the computations done by a party as part of a session" [13, Section 3.1].

Contributions In this paper we show that contrary to the claims in [2,4,3,11,12], the Ephemeral Key Reveal query is weaker than the Session-state Reveal query. Consequently, it follows that the CK and eCK models are incomparable, as the CK model does not allow compromise of the ephemeral key of the tested session. We show that the difference between the queries not only has theoretical but also practical implications, by providing two attacks on the NAXOS protocol, which can be performed using Session-state Reveal, but cannot be performed by using Ephemeral Key Reveal. The security model we use is nearly identical to the eCK model: we only replace Ephemeral Key Reveal by Session-state Reveal. Our attacks are also valid in the CK model, which implies that there is a meaningful difference between CK and eCK, as NAXOS was proven correct in the eCK model. We show how our attacks can be extended to the KEA, KEA+, and KEA+C protocols, and we discuss the interaction between Session-state Reveal and protocol transformations in e. g. the CK model.

The attacks presented here were found automatically by the Scyther tool [14]. For our attacks we use the NAXOS protocol exactly as specified in [1,2]. We assume that the protocol is implemented such that when a participant in the NAXOS protocol computes $H_2(x)$, where H_2 is a particular hash function in the NAXOS protocol, then x is in the local state just before the computation. As a result, performing a Session-state Reveal query just before the computation of $H_2(x)$ reveals x. This assumption does not require changing to the protocol. Rather, we make the contents of the session state explicit, as would be required for a proof in the CK model.

We proceed as follows. In Section 2 we explain some notation, and present the NAXOS protocol. Then, in Section 3 we show two attacks on this protocol that use Session-state Reveal. Further issues are discussed in Section 4, and we conclude in Section 5.

2 The NAXOS Key Exchange Protocol

The NAXOS protocol, as defined in [2,1], is shown in Figure 1. NAXOS builds on earlier ideas from the KEA and KEA+ protocols [15,16]. The purpose of the NAXOS protocol is to establish a shared symmetric key between two parties. Both parties have a long-term private key, e. g. sk_{a}, and initially know the public key of all other participants, e. g. pk_{b}. In Table 1 we give an overview of the notation used in the protocol as well as the remainder of this paper. We follow the notation from [1] where possible.

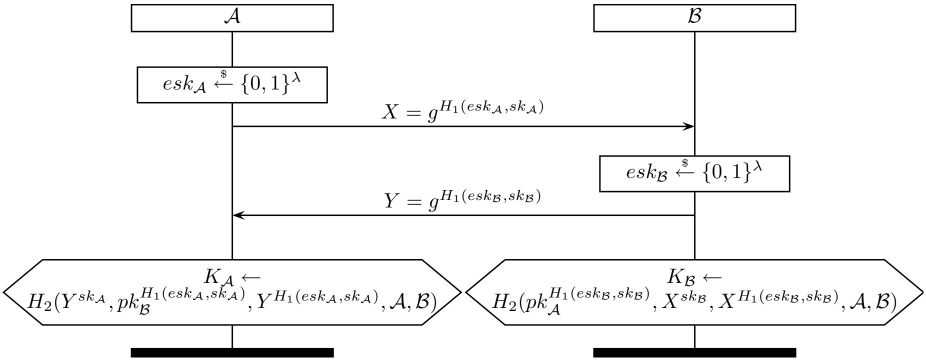

$\mathcal{A}$
$\mathcal{B}$
$esk_{\mathcal{A}} \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$
$X = g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}$
$esk_{\mathcal{B}} \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$
$Y = g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}$
$K_{\mathcal{A}} \leftarrow H_2(Y^{sk_{\mathcal{A}}}, pk_{\mathcal{B}}^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, Y^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, \mathcal{A}, \mathcal{B})$
$K_{\mathcal{B}} \leftarrow H_2(pk_{\mathcal{A}}^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, X^{sk_{\mathcal{B}}}, X^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, \mathcal{A}, \mathcal{B})$

Fig. 1. The NAXOS protocol. At the end of a normal execution we have that $K_{\mathcal{A}} = K_{\mathcal{B}}$ $(pk_x = g^{sk_x})$.

Table 1. Notation

$\mathcal{A}, \mathcal{B}$	The *initiator* and *responder* roles of the protocol.
a, b	Agents (participants) executing roles of the protocol.
G	A mathematical group of known prime order q.
g	A generator of the group G.
sk_{a}	The long-term private key of the agent a, where $sk_{\mathsf{a}} \in \mathbb{Z}_q$.
pk_{a}	The long-term public key of the agent a, where $pk_{\mathsf{a}} = g^{sk_{\mathsf{a}}}$.
H_1, H_2	Hash functions, where $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q$ and $H_2 : \{0,1\}^* \rightarrow \{0,1\}^{\lambda}$ (for some constant λ).
$esk_{\mathsf{a}}, esk'_{\mathsf{a}}$	Two different ephemeral keys of the agent a, generated in different sessions.
$\circ$	Written in place of a (bigger) term that is not relevant for the explanation at that point.
λ	A constant.
$x \xleftarrow{\$} S$	The variable x is drawn uniformly from the set S.
$x \leftarrow e$	The variable x is assigned the result of the expression e.

The protocol is designed to be secure in a very strong sense: the adversary is assumed to have the capability of learning long-term private keys, and also has the capability of learning short term data generated during a protocol session that does not include the private key.

The intuition behind the design of the protocol is that by combining the long-term private key with the short term ephemeral key inside the hash function, the adversary would need to have both of these elements to construct an attack. For example, the protocol should be secure if the adversary either (a) learns the long-term key of a participant during a session, or (b) learns the short-term data (except for the long-term key) of a participant during a session. A typical scenario for (b) is that the participant stores the long-term key on a TPM, and computes other operations in unprotected memory. For full details we refer the reader to [1,2].

In a normal execution, we have the following equivalences based on the properties of the modular exponentiation:

$$\begin{aligned}
X^{sk_{\mathcal{B}}} &= g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}}) sk_{\mathcal{B}}} &&= pk_{\mathcal{B}}^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})} && (1)\\
Y^{sk_{\mathcal{A}}} &= g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}}) sk_{\mathcal{A}}} &&= pk_{\mathcal{A}}^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})} && (2)\\
Y^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})} &= g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}}) H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})} &&= X^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})} && (3)
\end{aligned}$$

Hence, at the end of a normal protocol execution, the session key is computed by both parties as

$$H_2(g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}}) sk_{\mathcal{A}}}, g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}}) sk_{\mathcal{B}}}, g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}}) H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, \mathcal{A}, \mathcal{B}). \qquad (4)$$

3 Attacking NAXOS Using Session-state Reveal

3.1 Security Model eCK'

We use a slightly modified security model from the one defined in [1]. The only change is that we replace the Ephemeral Key Reveal query by the Session-state

Reveal query throughout the security definition. In particular, we require that whenever $H_2(x_1, \ldots, x_n)$ is computed, $x_1, \ldots, x_n$ are part of the local state just before the computation, and can therefore be revealed by a Session-state Reveal query. An example of an execution model where this condition holds, is a TPM setting in which $H_2(x_1, \ldots, x_n)$ is computed in local memory, whereas all other computations (such as $H_1(x)$ and g^x) are performed inside the TPM.

In contrast, applying the Ephemeral Key Reveal query to a session of the agent a in the eCK model (and original NAXOS proof) from [1] reveals only the ephemeral key esk_{a}.

Participants can perform roles of the protocol (such as initiator, $\mathcal{A}$, or responder, $\mathcal{B}$) multiple times, with various other partners. A single role instance performed by a participant is called a *session.*

Definition 1 (Session identifier). *The session identifier of a session sid is defined as the tuple* $(role, ID, ID{*}, comm_1, \ldots, comm_n)$*, where role is the role performed by the session (here initiator or responder), ID is the name of the participant executing sid, ID∗ the name of the intended communication partner, and* $comm_1, \ldots, comm_n$ *the list of messages that were sent and received.*

Definition 2 (Matching sessions for two-party protocols). *For a two-party protocol, sessions sid and sid′ are said to match if and only if there exist roles* $role, role'$ $(role \neq role')$*, participants* ID, ID'*, and message list* $L = comm_1, \ldots, comm_n$*, such that the session identifier of sid is* $(role, ID, ID', L)$ *and the session identifier of sid′ is* $(role', ID', ID, L)$*.*

In the eCK model, the adversary does not have access to a Session-state Reveal query, but instead has Ephemeral Key Reveal. Below we redefine the notion of clean and the security experiment from the eCK model [1, p. 8-9], in which we replace Ephemeral Key Reveal with Session-state Reveal, to define our security model eCK'.

Definition 3 (*clean* for eCK'). *In an AKE experiment (e. g. as defined in Definition 4 below), let sid be a completed AKE session performed by* a*, supposedly with some party* b*. Then sid is said to be* clean *if all of the following conditions hold:*

1. a *and* b *are not adversary-controlled (the adversary does not choose or reveal both the long-term and ephemeral keys of the participant and performs on behalf of the participant.)*
2. *The experiment does not include* Reveal(*sid*)*, i. e. the session key of session sid is not revealed.*
3. *The experiment does not include both* Long-term Key Reveal(a) *and* Session-state Reveal(*sid*)*.*
4. *If no session exists that matches sid, then the experiment does not include* Long-term Key Reveal(b)*.*
5. *If a session sid∗ exists*[1] *that matches sid, then*

[1] There may not be a unique matching session *sid*∗ for all executions of all protocols, but in the case of NAXOS, where each sent message contains randomness from the sending session, the matching session is unique if *sid* is a completed session.

(a) *the experiment does not include* Reveal($sid*$), *i. e. the session key of session* $sid*$ *is not revealed, and*
(b) *the experiment does not include both* Long-term Key Reveal(b) *and* Session-state Reveal($sid*$).

In the eCK' security model, queries such as Session-state Reveal may not be performed on *clean* sessions or their matching sessions as in [1, p. 7-8]. This is meant to exclude the cases in which an Session-state Reveal query trivially reveals the session key, such that no protocol could satisfy the security definition.

Definition 4 (AKE security experiment for eCK'). *In the eCK' AKE security experiment, the following steps are allowed:*

1. *The adversary may perform* Send(a, b, $comm$), Long-term Key Reveal(a), *and* Reveal(sid) *queries as in [1].*
2. *The adversary may perform a* Session-state Reveal(sid) *query. (This query replaces* Ephemeral Key Reveal(sid) *in the definition from [1].)*
3. *The adversary performs a* Test(sid) *query on a single* clean *session sid. A coin is flipped:* $b \xleftarrow{\$} \{0, 1\}$. *If* $b = 0$, *the test query returns a random bit string. If* $b = 1$, *the query returns the session key of sid. This query can be performed only once.*
4. *The adversary outputs a* Guess(b') *bit* b', *after which the experiment ends.*

An adversary $\mathcal{M}$ wins *the experiment if the* Guess(b) *bit* b *is equal to the bit* b' *from the* Test(b') *query.*

Definition 5 (eCK' security). *The advantage of the adversary* $\mathcal{M}$ *in the eCK' AKE experiment with AKE protocol* Π *is defined as*

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

We say that an AKE protocol is secure in the eCK' model if matching sessions compute the same session keys and no efficient adversary $\mathcal{M}$ *has more than a negligible advantage in winning the above experiment.*

We show two attacks on NAXOS in the eCK' model: One using test queries on sessions of the initiator type $\mathcal{A}$ and one using the responder type $\mathcal{B}$.

3.2 Attacking the Initiator

In Figure 2, we show an attack for a test query on an initiator session of NAXOS. The attack requires an active adversary that can reveal the local state of an agent.

The adversary can compute K_{a} on the basis of the revealed information (based on the algebraic properties of the group exponentiation, which are required for the core of the protocol).

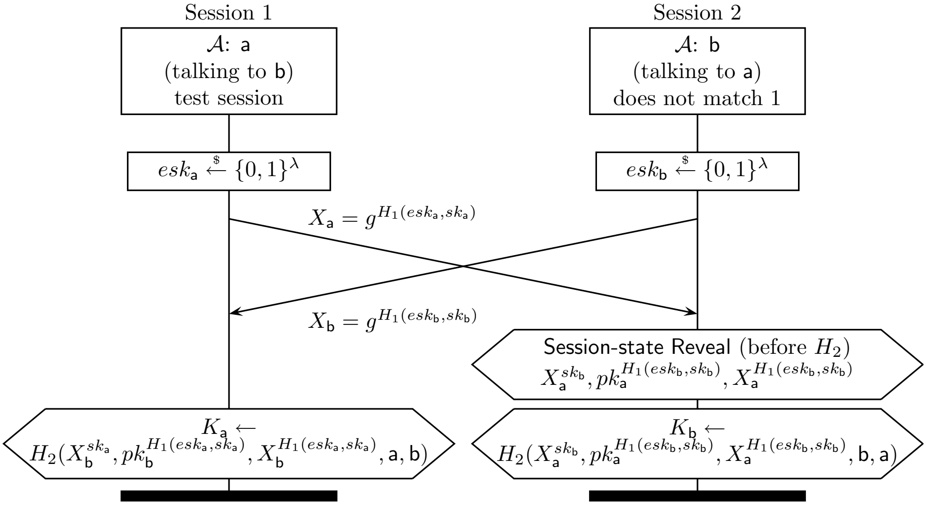

Session 1
$\mathcal{A}$: a
(talking to b)
test session
$esk_{\mathsf{a}} \xleftarrow{\$} \{0,1\}^{\lambda}$
Session 2
$\mathcal{A}$: b
(talking to a)
does not match 1
$esk_{\mathsf{b}} \xleftarrow{\$} \{0,1\}^{\lambda}$
$X_{\mathsf{a}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}$
$X_{\mathsf{b}} = g^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}$
Session-state Reveal (before H_2)
$X_{\mathsf{a}}^{sk_{\mathsf{b}}}, pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}$
$K_{\mathsf{a}} \leftarrow H_2(X_{\mathsf{b}}^{sk_{\mathsf{a}}}, pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{a}, \mathsf{b})$
$K_{\mathsf{b}} \leftarrow H_2(X_{\mathsf{a}}^{sk_{\mathsf{b}}}, pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, \mathsf{b}, \mathsf{a})$

Fig. 2. Attacking an initiator session. Note that $K_{\mathsf{a}} \neq K_{\mathsf{b}}$. The adversary can compute K_{a} after compromising the local state of b.

The attack proceeds as follows.

1. a starts an initiator instance, wanting to communicate with b.
2. a chooses her ephemeral key esk_{a}, and sends out $X_{\mathsf{a}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}$. The adversary learns this message.
3. b also starts an initiator instance, wanting to communicate with a.
4. b chooses her ephemeral key esk_{b}, and sends out $X_{\mathsf{b}} = g^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}$. The adversary learns this message.
5. The adversary sends the message X_{b} to a.
6. a computes the session key
$$K_{\mathsf{a}} = H_2(X_{\mathsf{b}}^{sk_{\mathsf{a}}}, pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{a}, \mathsf{b}). \tag{5}$$
7. The adversary sends the message X_{a} to b.
8. b computes the session key
$$K_{\mathsf{b}} = H_2(X_{\mathsf{a}}^{sk_{\mathsf{b}}}, pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}, \mathsf{b}, \mathsf{a}). \tag{6}$$
During the computation of K_{b}, the adversary uses Session-state Reveal to learn the input to H_2. In particular, the adversary learns $X_{\mathsf{a}}^{sk_{\mathsf{b}}}$, $pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}$, and $X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})}$.
9. The adversary now knows
$$pk_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})} = g^{sk_{\mathsf{a}} H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})} = X_{\mathsf{b}}^{sk_{\mathsf{a}}}, \tag{7}$$
$$X_{\mathsf{a}}^{sk_{\mathsf{b}}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}}) sk_{\mathsf{b}}} = pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \tag{8}$$
$$X_{\mathsf{a}}^{H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}}) H_1(esk_{\mathsf{b}}, sk_{\mathsf{b}})} = X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}. \tag{9}$$
The three terms on the right-hand side are the first three components of the session key K_{a} from Formula 5.

10. The adversary combines the elements with the names a and b, and applies H_2, resulting in K_{a}.

The above sequence of actions forms an attack on the protocol, because the adversary can learn the session key of the initiator a by revealing the local state of the second session. Furthermore, the test session is *clean* according to Definition 3 on page 24 because (1) neither a nor b are adversary-controlled, (2) no Reveal queries are performed, (3) no long-term keys are revealed, and (4) session 2 is not a partner to the test session 1. Therefore, the attack violates security in the eCK' model.

Some further observations regarding this attack:

- The sessions compute different session keys: $K_{\mathsf{a}} \neq K_{\mathsf{b}}$, because the order of the participant names a, b is reversed.
- The adversary does not need to learn any ephemeral keys for this attack.
- Even in other existing interpretations of the partner function (or freshness) from literature (matching conversations, external session identifiers, explicit session identifiers, etc.) the two sessions are not partners. Consequently, the NAXOS protocol is therefore also not secure in other models that allow Session-state Reveal, such as the CK model [5].

3.3 Attacking the Responder

Second, we show an attack for a test query on a responder session in Figure 3. It seems this attack is more easily exploited than the previous one.

The attack proceeds as follows.

1. The adversary chooses an arbitrary bit string κ.
2. The adversary computes g^{κ} and sends the result to a responder instance of a, with sender address b.
3. a receives the message and assigns $X_{\mathsf{b}} = g^{\kappa}$.
4. a chooses her ephemeral key esk_{a}, and sends out $X_{\mathsf{a}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}$. The adversary learns this message.
5. a computes the session key

$$K_{\mathsf{a}} = H_2(pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, X_{\mathsf{b}}^{sk_{\mathsf{a}}}, X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{b}, \mathsf{a}) \tag{10}$$

which is equal to

$$H_2(pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, g^{\kappa\ sk_{\mathsf{a}}}, g^{\kappa\ H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{b}, \mathsf{a}). \tag{11}$$

6. The adversary redirects X_{a} to a responder instance of b. The adversary can insert an arbitrary participant name in the sender field of the message, which b takes to be the origin of the message.
7. b computes his ephemeral secret, combines it with his long term key, and sends out the corresponding message.
8. b computes his session key K_{b} (which differs from K_{a}). Before applying H_2, b computes the second component $X_{\mathsf{a}}^{sk_{\mathsf{b}}}$.

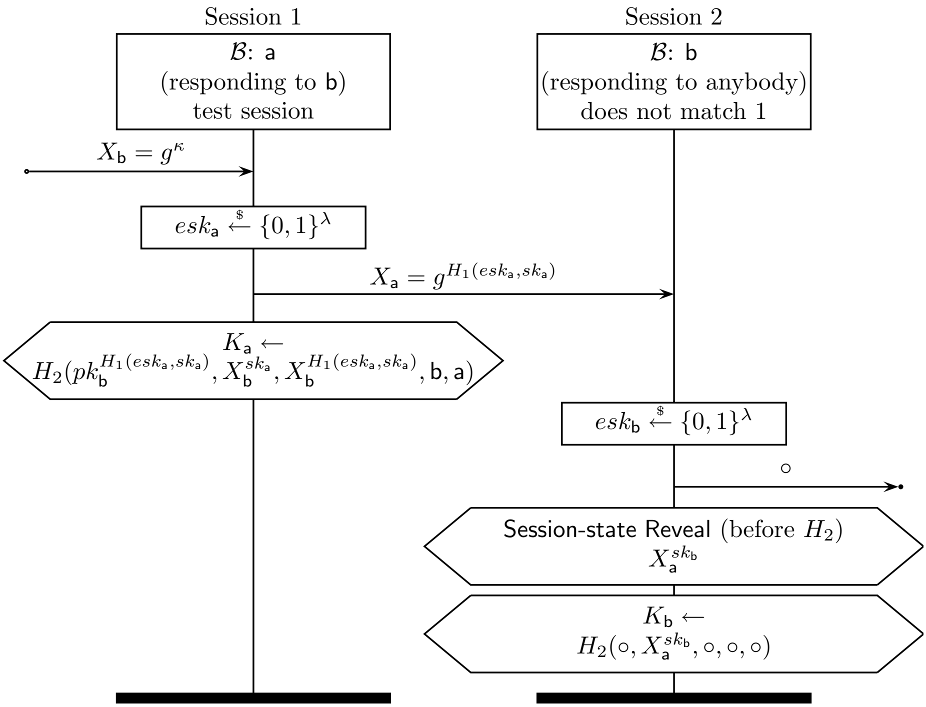

Session 1
$\mathcal{B}$: a
(responding to b)
test session
Session 2
$\mathcal{B}$: b
(responding to anybody)
does not match 1
$X_{\mathsf{b}} = g^{\kappa}$
$esk_{\mathsf{a}} \xleftarrow{\$} \{0,1\}^{\lambda}$
$X_{\mathsf{a}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}$
$K_{\mathsf{a}} \leftarrow$
$H_2(pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, X_{\mathsf{b}}^{sk_{\mathsf{a}}}, X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \mathsf{b}, \mathsf{a})$
$esk_{\mathsf{b}} \xleftarrow{\$} \{0,1\}^{\lambda}$
$\circ$
Session-state Reveal (before H_2)
$X_{\mathsf{a}}^{sk_{\mathsf{b}}}$
$K_{\mathsf{b}} \leftarrow$
$H_2(\circ, X_{\mathsf{a}}^{sk_{\mathsf{b}}}, \circ, \circ, \circ)$

Fig. 3. Attack on a responder session. We have $K_{\mathsf{a}} \neq K_{\mathsf{b}}$. The adversary can compute (and even contribute to) K_{a} after revealing the local state of b.

9. The adversary uses Session-state Reveal on the session of b directly before the application of H_2 to learn $X_{\mathsf{a}}^{sk_{\mathsf{b}}}$.
10. The adversary knows κ, X_{a}, and $X_{\mathsf{a}}^{sk_{\mathsf{b}}}$. Furthermore, as the public keys are public, the adversary also knows pk_{a}. Hence the adversary also knows, or can compute:

$$X_{\mathsf{a}}^{sk_{\mathsf{b}}} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}}) sk_{\mathsf{b}}} = pk_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}, \tag{12}$$

$$(pk_{\mathsf{a}})^{\kappa} = g^{sk_{\mathsf{a}} \kappa} = X_{\mathsf{b}}^{sk_{\mathsf{a}}}, \tag{13}$$

$$(X_{\mathsf{a}})^{\kappa} = g^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}}) \kappa} = X_{\mathsf{b}}^{H_1(esk_{\mathsf{a}}, sk_{\mathsf{a}})}. \tag{14}$$

The three terms on the right-hand side are the first three components of the session key K_{a} from Formula 10.
11. The adversary combines the elements and applies H_2, resulting in K_{a}.

This sequence forms an attack on the protocol, because the adversary can use data revealed from session 2 in order to compute the session key of the test session 1. The test session is also clean according to Definition 3 on page 24. In practical terms, this attack even allows the adversary to determine a part of the session key of a.

For this attack there are also some observations to be made:

- The responder session of b is not a partner to the session of a in terms of matching sessions. Also, in other partner existing interpretations from literature (external session identifiers, explicit session identifiers, etc.) they would also not match.
- The adversary chooses κ, and can therefore influence the session key.
- In this attack, the adversary does not need to learn any long term private keys or ephemeral keys.
- The attack is also valid in the CK model: the sessions are not partners for a number of reasons, for example because their choice of agents differs. Session 1 has $\{\mathsf{a}, \mathsf{b}\}$ and session 2 has $\{\mathsf{b}, z\}$ where z is an arbitrary participant. Hence the adversary can choose $z \neq \mathsf{a}$.

4 Discussion

The structure of our attacks can be used to attack some protocols that were proven correct in the CK model [5]. We first briefly discuss these other protocols, and afterwards discuss the implications for existing proofs and protocol transformation theorems in the CK model.

4.1 The KEA, KEA+ and KEA+C Protocols

In [16], the KEA+ protocol is proven correct in the CK model from [5]. KEA+ can be viewed as a predecessor of the NAXOS protocol, and uses a similar setup. Two other variants of this protocol are KEA+C from [16] and KEA from [15]. All three protocols compute the session key using a hash function, which takes as inputs components built by the communication partners. Each of the crucial inputs is a modular exponentiation that includes in the exponent both randomness and the long-term private keys of one of the participants. The proof in [16] of the security of KEA+ in the CK model assumes that Session-state Reveal is defined such that only the ephemeral keys are revealed.

The attacks presented in this paper on the NAXOS protocol also work within the CK model for the KEA, KEA+, and KEA+C protocols after minimal modifications. The attacks use the exact same scenarios and exploit the same Session-state Reveal definition, in which the inputs to the final hash function are part of the local state.

4.2 Session-state Reveal and Protocol Transformations in the CK Model

In the CK model [5] the Session-state Reveal query is defined as revealing the full internal state of the Turing machine executing the protocol to the adversary. This internal state is not defined within the CK model, and can be viewed as a parameter of the correctness proof of the protocol. As a result, proving that a protocol is correct in the CK model requires one to define this internal state.

Technically this implies that the resulting proof holds only for execution models corresponding to this definition of the internal state. This puts restrictions on the implementation details of the protocol code as well as on the platform executing the code.

In existing protocol proofs in the CK model the internal state is not explicitly stated as a parameter of the correctness proof. In most cases, the internal state is simply defined as the ephemeral secret, i. e. the private exponent of a participant in a Diffie-Hellman style key exchange. As a consequence, any implementation in which the local state (as revealed to an adversary) contains more information, falls outside of the scope of the proof.

However, making the definition of the internal state an explicit parameter of the correctness proofs has implications for the methodology underlying the CK model. Central to the methodology underlying the model is the notion of security preserving (or security enhancing) protocol transformations. An example of a central result is Theorem 6 from [5, p. 16]. This theorem involves the notion of (MT-)authenticators, which are protocol transformations that satisfy certain security preserving/enhancing properties. The theorem aims to establish that a protocol that is secure in one security model (AM) can be transformed by an authenticator into another protocol that satisfies the same security property in a stronger adversarial model (UM). In this case the security property is SK-security, which involves an adversary that has access to the Session-state Reveal query.

Theorem 6 from [5, p. 16] can be rephrased in the following way. Let P be a protocol and let f be an (MT-) authenticator. If the protocol P satisfies SK-security in the AM model, then the protocol $f(P)$ satisfies SK-security in the UM model. The proof of this theorem is generic and applies to any authenticator.

In the precondition of this theorem and the definition of authenticators, there is no constraint that prevents authenticators from changing the local state, i. e., there is no requirement on f that ensures that Session-state Reveal for P is equal to Session-state Reveal for $f(P)$. Note that from a practical point of view, including such a requirement may be unrealistic: transforming the protocol in any non-trivial way implies that the local state of the protocol $f(P)$ is different from that of the protocol P.

If we assume that that applying the authenticator does change the local state, i. e. Session-state Reveal for $f(P)$ is not equal to Session-state Reveal for P, then it is not immediately clear how to prove the correctness of the theorem, as it would involve proving that the transformation of the local state does not introduce new attacks, possibly along the lines of the attacks presented here, that exploit Session-state Reveal for $f(P)$. Recall that the attacks on NAXOS do not require revealing the ephemeral key (which would already be in the local state of P), nor the long-term private keys (which would be excluded from Session-state Reveal for $f(P)$), but rather some intermediate computations. We expect that a generic proof of this theorem requires a significant restriction on the class of allowed authenticators.

The existence of these attacks shows the importance of explicitly specifying the definition of local state as it is used in a proof: e. g. KEA+ is not secure in the CK model if the inputs to the final hash function are part of the local state.

It would be more precise to say that in [16] KEA+ is proven secure with respect to the CK model if the local state only includes the ephemeral keys.

5 Conclusion

In common definitions of AKE security the Session-state Reveal query is underspecified. The definition of Session-state Reveal is only made explicit in particular protocol proofs. This approach turns the exact definition of Session-state Reveal into a parameter of the exact security provided by the protocol. As a result, stating that two protocols are provably secure in e. g. the CK model does not mean they meet exactly the same property.

In [1,2] the Session-state Reveal query is replaced by the Ephemeral Key Reveal query, which is claimed to be at least as strong as Session-state Reveal. Thus, the notion of Session-state Reveal is reduced to Ephemeral Key Reveal. Reducing Session-state Reveal to Ephemeral Key Reveal simplifies proofs significantly: one does not need to define what exactly is part of the ephemeral key, but one only needs to prove that no information about the ephemeral key is revealed [1,4,3]. However, the validity of this reduction has not been proven.

The validity of the reduction is informally argued in [2], and similar arguments can be found in other works that use the eCK model [4,3], e. g. in [4, p. 333]: "In general, by specifying that the session specific private information (the session state) is part of the ephemeral private key, the Session-state Reveal and Ephemeral Key Reveal queries can be made functionally equivalent".

In this paper we have shown that the reduction is invalid, that is, a security model with Ephemeral Key Reveal (eCK) is not as strong as a model with Session-state Reveal (eCK'). The attacks presented here on the NAXOS protocol, which was proven correct for Ephemeral Key Reveal in [1], strictly depend on the use of the Session-state Reveal query.

The attacks presented here fall just outside the eCK security model, and they therefore do not indicate a problem with the proofs in [1]. Instead, what the attacks indicate is that the eCK security model, and similarly the property that is proved correct, is not as strong as suggested in e. g. [1]. Furthermore, the attacks are also valid in the CK model, which shows that the difference between CK and eCK is in fact meaningful in practice. In particular, we have shown that one can prove real protocols secure in eCK which are not secure in CK, and are vulnerable to attacks where the local state is revealed. Consequently, the CK and eCK models are not only theoretically, but also practically incomparable.

The structure of our attacks on NAXOS can be translated to attacks on the KEA, KEA+, and KEA+C protocols from [15,16]. As a result, also these protocols are not CK-secure if the session state includes the inputs to the final hash function. We furthermore observed that it is non-trivial to combine protocol proofs that consider the Session-state Reveal query, such as those in the CK model, with protocol transformations.

The idea behind the NAXOS protocol (which is already found in KEA and KEA+) is appealing: by strongly connecting the long- and short-term information, the adversary would be required to know both elements to perform an attack. However, in order to use the combination of these elements securely in the protocol, in particular for transmission, there are further computations needed. These subsequent computations often influence the local state. This effect is not captured by the definition of Ephemeral Key Reveal, which is the ultimate problem with the reduction from Session-state Reveal to Ephemeral Key Reveal, as was already noted in [13]. The attacks presented in this paper exploit exactly this difference.

A possible practical interpretation of the difference between the models is the following. The CK model considers a TPM implementation, where parts of the protocol are computed in unprotected memory, specified by the contents of the session-state, but the long-term private keys are protected by the TPM. The adversary may be able to learn the contents of the unprotected memory at some point, but not necessarily all the time. In contrast, the eCK model considers a malicious (i. e. predictable or information-leaking) random number generator, which implies that the adversary learns all ephemeral keys.

The question remains whether it is possible to adapt NAXOS to satisfy a security model similar to eCK that allows for Session-state Reveal queries.

References

1. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007)
2. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. Cryptology ePrint Archive, Report 2006/073 (2006), http://eprint.iacr.org/
3. Okamoto, T.: Authenticated key exchange and key encapsulation in the standard model. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 474–484. Springer, Heidelberg (2007)
4. Ustaoglu, B.: Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. Des. Codes Cryptography 46(3), 329–342 (2008)
5. 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)
6. Krawczyk, H.: HMQV: A high-performance secure diffie-hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
7. Bresson, E., Chevassut, O., Pointcheval, D., Quisquater, J.: Provably authenticated group Diffie-Hellman key exchange. In: CCS 2001: Proceedings of the 8th ACM conference on Computer and Communications Security, pp. 255–264. ACM Press, New York (2001)
8. Menezes, A., Ustaoglu, B.: Comparing the pre- and post-specified peer models for key agreement. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 53–68. Springer, Heidelberg (2008)
9. Choo, K.K., Boyd, C., Hitchcock, Y.: Examining indistinguishability-based proof models for key establishment proofs. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 624–643. Springer, Heidelberg (2005)

10. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
11. Xia, J., Wang, J., Fang, L., Ren, Y., Bian, S.: Formal proof of relative strengths of security between ECK 2007 model and other proof models for key agreement protocols. Cryptology ePrint Archive, Report 2008/479 (2008) http://eprint.iacr.org/ (retrieved January 12, 2009)
12. Lee, J., Park, C.: An efficient authenticated key exchange protocol with a tight security reduction. Cryptology ePrint Archive, Report 2008/345 (2008) http://eprint.iacr.org/ (retrieved January 12, 2009)
13. Boyd, C., Cliff, Y., Nieto, J., Paterson, K.: Efficient one-round key exchange in the standard model. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 69–83. Springer, Heidelberg (2008)
14. Cremers, C.: The Scyther Tool: Verification, falsification, and analysis of security protocols. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 414–418. Springer, Heidelberg (2008)
15. NIST: SKIPJACK and KEA algorithm specification (1998), http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf
16. Lauter, K., Mityagin, A.: Security analysis of KEA authenticated key exchange protocol. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 378–394. Springer, Heidelberg (2006)

Secure Pairing of "Interface-Constrained" Devices Resistant against Rushing User Behavior

Nitesh Saxena and Md. Borhan Uddin

Computer Science and Engineering
Polytechnic Institute of New York University
nsaxena@poly.edu, borhan@cis.poly.edu

Abstract. "Secure Device Pairing" is the process of bootstrapping secure communication between two devices over a short- or medium-range wireless channel (such as Bluetooth, WiFi). The devices in such a scenario can neither be assumed to have a prior context with each other nor do they share a common trusted authority. Fortunately, the devices can generally be connected using auxiliary physical channel(s) (such as audio, visual, tactile) that can be authenticated by the device user(s), thus forming the basis for pairing. However, lack of good quality output interfaces (e.g, a speaker, display) and/or receivers (e.g., microphone, camera) on certain devices makes pairing a very challenging problem in practice.

We consider the problem of "*rushing user*" behavior in device pairing. A rushing user is defined as a user who in a rush to connect her devices, would skip through the pairing process, if possible. Most prior pairing methods, in which the user decides the final outcome of pairing, are vulnerable to rushing user behavior – the user can simply "accept" the pairing, without having to correctly take part in the decision process. In this paper, we concentrate on most common pairing scenarios (such as pairing of a WiFi laptop and an access point), whereby one device (access point) is constrained in terms output interfaces, while the other (laptop) has a decent quality output interface but no receiver. We present the design and usability analysis of two novel pairing methods, which are resistant to a rushing user and require only minimal device interfaces on the constrained device. One of the most appealing applications of our proposal is in defending against common threat of "Evil Twin" attacks in public places (e.g, cyber-cafes, airport lounges).

Keywords: Device Pairing, Authentication, Usability, Security, Evil Twin Attacks, Wireless Communication.

1 Introduction

Short-range wireless communication, based on technologies such as Bluetooth and WiFi, is becoming increasingly popular and promises to remain so in the future. With this surge in popularity, come various security risks. Wireless communication channel is easy to eavesdrop upon and to manipulate, and therefore a fundamental security objective is to secure this communication channel. In this paper, we will use the term "pairing" to refer to the operation of bootstrapping secure communication between two devices connected with a short-range wireless channel. The examples of pairing, from

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 34–52, 2009.

day-to-day life, include pairing of a WiFi laptop and an access point, a Bluetooth keyboard and a desktop. Pairing would be easy to achieve, if there existed a global infrastructure enabling devices to share an on- or off-line trusted third party, a certification authority, a PKI or any pre-configured secrets. However, such a global infrastructure is close to impossible to come by in practice, thereby making pairing an interesting and a challenging real-world research problem. The problem has been at the forefront of various recent standardization activities, see [24].

A promising and well-established research direction to pairing is to use an auxiliary physically authenticatable channel, i.e., physical channel, also called an out-of-band (OOB) channel, which is governed by humans, i.e., by the users operating the devices. Examples of OOB channels include audio, visual, tactile channels. Unlike the wireless channel, on the OOB channel, an adversary is assumed to be incapable of modifying messages, however, it can eavesdrop on, delay, drop and replay them. A pairing method should therefore be secure against such an adversary.

The usability of a pairing method based on OOB channels is clearly of utmost importance. In pairing scenarios (such as pairing of a WiFi laptop and an access point) involving devices that lack good quality output interfaces (e.g, a speaker, display) and/or receivers (e.g., microphone, camera), establishing OOB channels is quite difficult. Minimizing the user burden in pairing such "interface-constrained" devices is thus a very challenging problem. Since the OOB channels typically have low bandwidth, the shorter the data that a pairing method needs to transmit over these channels, the better the method becomes in terms of usability.

Various pairing protocols have been proposed so far. These protocols are generally based on the bidirectional automated device-to-device (d2d) OOB channels. Such d2d channels require both devices to have transmitters and the corresponding receivers. In settings, where d2d channel(s) do not exist (i.e., when at least one device does not have a receiver) and even otherwise, same protocols can be based upon device-to-human (d2h) and human-to-device (h2d) channel(s) instead. Depending upon the protocol, only two d2h channels might be sufficient, such as in case when the user has to perform a very simple operation (such as "comparison") of the data received over these channels. Clearly, the usability of d2h and h2d channel establishment is even more critical than that of a d2d channel.

The earlier pairing protocols [4], [14] require at least 80 to 160 bits of data to be transmitted over the OOB channels. The more recent, so-called SAS- (Short Authenticated Strings) based protocols, [12] and [15], reduce the length of data to be transmitted over the OOB channels to only 15 bits or so, for a reasonable level of security.[1]

Based on the above-mentioned protocols, a number of pairing methods with various OOB channels have been proposed. All prior pairing methods are reviewed in the following section of the paper.

In this paper, we consider the problem of "*rushing user*" behavior in device pairing. We define a rushing user as a user who in a rush to connect her two devices, tends to skip through the pairing process, if possible. Such a rushing user behavior does exist in practice and in fact, is quite common. It has been shown that computer users tend

[1] The concept of SAS-based authentication was first introduced by Cagalj et al. [27], followed by Vaudenay [26]. MANA protocols [7] addressed a similar problem.

to be “task-focussed” [6]. For example, in the context of phishing attacks [6], when a user logs on to the website of her bank, her focus is (e.g.,) to pay a bill which is past due; she would tend to ignore any warning indicating a phishing attempt. Similarly, in the context of device pairing, when a user wants to connect her Bluetooth laptop with her cell phone, her primary task is (e.g.,) to transfer her pictures or synchronize her calendar; when she wants to connect her Bluetooth cell phone with a headset, she is eager to speak to someone. The pairing process, from user’s perspective, is nothing but a hindrance in her intended task, and therefore she would quickly tend to skip through this process, if possible.

All previously proposed pairing methods can be broadly classified into two categories: (1) device-controlled (DC) method, where the OOB strings are transferred between two devices with the help of the user and eventually the devices decide the outcome of pairing, and (2) user-controlled (UC) method, where the user herself compares the OOB strings output by the devices and decides the pairing outcome. We observe that all UC pairing methods are vulnerable against a rushing user, whereas the DC methods are not. In the UC methods, the user can simply “accept” the pairing, without having to take part in the decision process correctly. On the other hand, in the DC methods, the user is somewhat forced to perform the pairing process correctly, because otherwise she will not be able to connect her two devices.

Our Contributions. In this paper, we concentrate on most common pairing scenarios, wherein one device is constrained in terms output interfaces, while the other has a decent quality output interface but no receiver. A common example of such a scenario is pairing of a WiFi laptop and an access point (the latter is a constrained device with no display or receiver; the former has a full display and keypad, but no receiver). We note that some prior work addresses the problem of pairing interface-constrained devices (e.g., [17][18]). However, this paper is the first, to the best of our knowledge, to address an even more challenging problem of pairing interface-constrained devices in a rushing user resistant manner. We present the design and usability analysis of two novel pairing methods, which are resistant to a rushing user and require only minimal device interfaces on the constrained device. The two proposed pairing methods, called “*color pairing*” and “*alphanumeric pairing*,” are based on two different types of output interfaces on the constrained device, i.e., a Multi-Color LED and a Sixteen Segment Display (SSD), respectively. Both of these interfaces are minimal in terms of their cost and size, are commonly available and thus can be easily added onto constrained devices (such as access points, printers).[2] In the color pairing method, the user is required to transfer colors displayed through the multi-color LED of the constrained device to the other device. In the alphanumeric pairing method, the user simply transfers the characters displayed by the SSD of the constrained device onto the other device. Clearly, both our methods are based on different sensory capabilities of human users and have different usability implications

Based on a usability study of the proposed pairing methods, we conclude that the color pairing method based on four distinct colors is quite suitable for most devices and

[2] The use of such interfaces is not just limited to the pairing operation. For example, a multi-color LED can serve the purpose of a general-purpose LED on an access point, and the SSD can serve as a minimal display on a printer.

pairing scenarios, as it turned out to be very efficient, robust to human errors and user-friendly. This method is ideal, as our testing indicates, for defending against a common threat of *evil twin* access points (e.g., in cyber-cafes, airport lounges), in a rushing user resistant manner.[3]

Organization. The rest of the paper is organized as follows. In Section 2, we review the prior pairing methods. In Section 3, we describe the security model and summarize relevant protocols. In Sections 4 and 5, we present the design and implementation of our color pairing and alphanumeric pairing methods. Finally, in Section 6, we discuss our experimental usability study with respect to the proposed pairing methods and the underlying results.

2 Related Work

In this section, we discuss prior pairing methods, their applicability to interface constrained devices and whether or not they are resistant to rushing user behavior. Recall a DC method is resistant to rushing user, whereas a UC method is not.

In their seminal work, Stajano, et al. [23] proposed establishing a shared secret between two devices using a link created through a physical contact (such as an electric cable). This is a DC method and is resistant to rushing user. However, in many settings, establishing such a physical contact might not be possible, for example, the devices might not have common interfaces to do so or it might be too cumbersome to carry the cables along. Balfanz, et al. [4] extended this approach through the use of infrared as a d2d channel – the devices exchange their public keys over the wireless channel followed by exchanging (at least 80-bit long) hashes of their respective public keys over infrared. This is also a DC method. The main drawback of this method, however, is that it is only applicable to devices equipped with infrared transceivers. Moreover, the infra-red channel is not easily perceptible by human users.

Another approach taken by a few research papers is to perform the key exchange over the wireless channel and authenticate it by requiring the users to manually and visually compare the established secret on both devices. Since manually comparing the established secret or its hash is cumbersome for the users, methods were designed to make this visualization simpler. These include Snowflake mechanism [9] by Levienet et al., Random Arts visual hash [16] by Perrig et al., etc. These methods require high-resolution displays and are thus only applicable to a limited number of devices, such as laptops. Moreover, these are UC methods and thus are vulnerable to a rushing user.

Based on the pairing protocol of Balfanz et al. [4], McCune et al. proposed the “Seeing-is-Believing” (SiB) method [13]. SiB involves establishing two unidirectional visual d2d channels – one device encodes the data into a two-dimensional barcode and the other device reads it using a photo camera. SiB is a DC method. However, since the method requires both devices to have cameras, it is only suitable for pairing devices such as camera phones.

Goodrich, et al. [10], proposed “Loud-and-Clear (L&C)”, a pairing method based on “MadLib” sentences. The main idea of L&C is to encode the OOB data into MadLib

[3] This only involves unidirectional authentication of the access point to the laptop.

sentences and have the user compare these sentences displayed or spoken out on two devices. Clearly, this is a UC method and is thus vulnerable to rushing user behavior. Moreover, the method is not applicable to pairing scenarios where one of the devices does not have a display or a speaker.

Saxena et al. [19] proposed a pairing method based on visual OOB channel. The method uses one of the SAS protocols [12], and is aimed at pairing two devices A and B (such as a cell phone and an access point), only one of which (say, B) has a relevant receiver (such as a camera). First, a unidirectional d2d channel is established by device A transmitting the SAS data, e.g., by using a blinking LED and device B receiving it using a video camera. This is followed by device B comparing the received data with its own copy of the SAS data, and transmitting the resulting bit of comparison over a d2h channel (say, displayed on its screen). Finally, the user reads this bit transmitted and accordingly indicates the result to device A by transmitting a bit over an h2d input channel. In one direction (i.e., from A to B), this is a DC method and is thus resistant to rushing user behavior. In the other direction, however, it is a UC method – a rushing user can simply accept the pairing on A without looking at the pairing outcome on B. It is important to note, however, that in case of any attack, device B will be "locked out" and will not allow any connection to and from device A (and it will detect any connection attempts from an attacking device).[4] This way the user will not be able to establish real communication with device B (e.g., transfer an image file from B to A), and will thus resort to repeating the pairing process. The pairing methods we propose in this paper utilize this unidirectional pairing approach of [19].

Uzun et al. [25] carry out a comparative usability study of simple pairing methods. They consider pairing scenarios where devices are capable of displaying 4-digits of SAS data. In what they call the "Compare-and-Confirm" approach (a UC method), the user simply reads and compares the SAS data displayed on both devices. The "Select-and-Confirm" approach (a DC method), on the other hand, requires the user to select a 4-digit string (out of a number of strings) on one device that matches with the 4-digit string on the other device. The third approach, called "Copy-and-Confirm" (a DC method), requires the user to read the data from one device and input it onto the other. Both Select-and-Confirm and Copy-and-Confirm are DC methods and therefore offer protection against a rushing user. However, these methods are only limited to devices (such as cell phones) which have good quality displays and keypads. Our alphanumeric pairing method is quite similar in flavor to the Copy-and-Confirm method of [25], however, it is applicable to interface-constrained devices. Kuo et al. [11] defined a common baseline for hardware features and a consistent, interoperable user experience across pairing of different devices. This work did not yield any pairing method as such.

Some recent papers have focused upon pairing devices which possess constrained interfaces. These include the BEDA method [21] which requires the users to transfer the SAS strings from one device to the other using "button presses". BEDA is based on the protocol of [19]. The constrained device encodes its SAS string into the time intervals between two consecutive blinkings of the (regular) LED, and as and when this

[4] In case of a pairing failure, device B can keep showing a warning to the user indicating that device A is possibly being connected to an attacker device, and ask the user to "re-pair" the two devices.

device blinks, the user presses a button on the other device. BEDA is a DC method and is therefore resistant to rushing user behavior. BEDA is also universally applicable to most pairing scenarios. However, as indicated in the results of [21], it requires about one minute to complete the pairing process, which might be too slow in practice. Moreover, the user needs to pay close attention to both devices simultaneously to maintain synchronization. This will be particularly hard when one of the devices is distant (e.g., a wall-mounted access point). The methods that we present in this paper are more efficient in comparison to BEDA, as we will see in the later sections of the paper.

In [17], Saxena et al. presented a pairing method universally applicable to any pair of devices. The method can be based on any of the existing SAS protocols and does not require devices to have good transmitters or any receivers, that is, just a pair of LEDs is sufficient. The method involves users comparing very simple audiovisual patterns, such as "beeping" and "blinking", transmitted as simultaneous streams which form two synchronized d2h channels. Most recently, the approach of [17] was extended by making use of an auxiliary device, such as a smartphone [20]. Both these methods, however, are UC methods and thus offer no protection against a rushing user. In an independent result [18], Roth et al. present a method similar to the "blinking" method presented in [17]. The method of [18] is aimed at the detection of evil twin access points. The two methods, however, differ significantly in their implementation and therefore in terms of user experience (see [17] for details regarding the differences). This method is also a UC method and thus offers no protection against a rushing user. The pairing methods we propose in this paper aptly address the problem of evil twin access points, however, unlike [18], our methods also offer resistance against rushing user behavior.

In [22], Soriente et al. consider the problem of pairing two devices which might not share any common wireless communication channel at the time of pairing, but do share only a common audio channel. This is a DC method, however, it is only limited to devices which possess a speaker at the transmitting end and a microphone at the receiving end.

3 Security Model and Applicable Protocols

The pairing protocols, on which our methods are built, are based upon the following communication and adversarial model [26]. The devices being paired are connected via two types of channels: (1) a short-range, high-bandwidth bidirectional wireless channel and (2) one or more auxiliary low-bandwidth physical OOB channel(s). Based on the type of devices being used, the OOB channel(s) can be device-to-device (d2d), device-to-human (d2h), or human-to-device (h2d). An adversary attacking the pairing protocol is assumed to have full control of the wireless channel, namely, he or she can eavesdrop, delay, drop, replay and modify messages. On the OOB channel, the adversary can eavesdrop, delay, drop, replay and re-order messages; however, it can not modify them. In other words, the OOB channel is assumed to be an authenticated channel.

The security notion applied to a pairing protocol in this setting is adopted from the model of authenticated key agreement by Canneti and Krawczyk [5]. In this model, a multi-party setting is considered wherein a number of parties simultaneously run multiple/parallel instances of pairing protocols. In practice, however, it is reasonable to assume that there are only two parties running just a few serial or parallel instances of the pairing protocol. For example, during the authentication of an ATM transaction there are only two parties, namely the ATM machine and a user. Further, the user is restricted to only three authentication attempts. The security model does not consider denial-of-service (DoS) attacks. Note that with a wireless channel explicit attempts to prevent protocol-level DoS attacks are not useful because an adversary can simply launch an attack by jamming the wireless signal.

To date, two three-round pairing protocols based on short authenticated strings (SAS) have been proposed: [15] and [12]. In a communication setting involving two users restricted to running three instances of the protocol these SAS protocols need to transmit only k $(= 15)$ bits of data over the OOB channel. As long as the cryptographic primitives used in the protocol are secure, an adversary attacking one of these protocols can not win with a probability significantly higher than 2^{-k} $(= 2^{-15})$. This gives us security equivalent to that provided by 5-digit PIN-based ATM authentication. The pairing methods proposed in this paper are based upon the SAS protocols mentioned above, with a variation presented in [19], as discussed in Section 2.

4 "Color Pairing" Using a Multi-Color LED

In this section, we discuss the design and implementation of a novel color-based pairing method, called "color pairing", which is resistant against rushing user. In our method, one device (denoted as A) is equipped with a "Multi-Color" LED [2] and the other device (denoted as B) has a display and a keypad. Device A encodes its SAS data into colors and displays each color one-by-one through the LED, the user reads each color and accordingly selects the corresponding color (from all possible displayed colors) on device B, maintaining synchronization (i.e., transition between two consecutive colors) of transmission and reception. In this manner, the SAS data is transferred from device A to device B, while maintaining synchronization. Once device B decodes the received SAS data, it compares it with its own local copy; and accordingly accepts or rejects the pairing instance. Notice that our method is a DC method and is thus resistant to rushing user behavior.

Clearly our method is based on the number of "human-distinguishable" colors an off-the-shelf multi-color LED is capable of displaying – the more the number of such colors, the more the number of SAS bits can be transmitted everytime. In the following subsections, we describe the selection of such human-distinguishable colors, the encoding of SAS data into these colors and rushing user resistant transmission and processing of SAS data.

4.1 Selection of "Human-Distinguishable" Colors

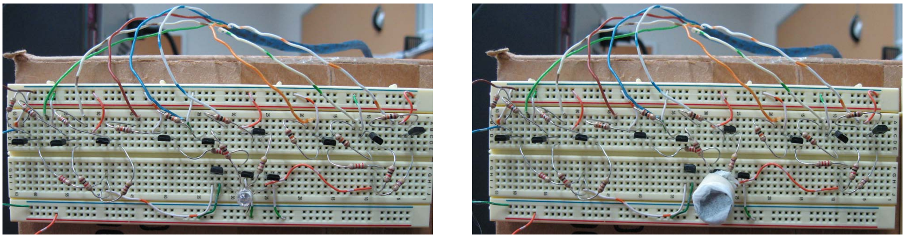
In the 24-bit RGB color model, there are 2^{24} (i.e., about 16-million) distinct colors. Out of these colors, our goal was to find out the colors which can be generated using a multi-color LED and which are easily and unambiguously distinguishable by human users without any prior training. One recent study [8] shows that there are only 11 non-conflicting colors identified for categorical images. In this set of 11 colors of [8], there seems to be some pairs of colors which do not appear that distinct, e.g., light green and dark green, and blue and dark blue. If a user is shown blue (without showing dark blue beforehand) and asked to identify whether it is dark blue or blue, it is highly likely that the user will be confused, because the user would not be sure whether a more or less bright version of the color exists or not. We generated the same set of 11 colors and some more colors on monitor screen using the simulated annealing method as described in [8]. However, with the help of some preliminary testing on some human users, we concluded that the number of distinct colors is definitely not more than 11; in fact, it might be less than 11. This was also because all the 11 colors of [8] were not easily distinguishable by human users on multi-color LED as light and dark shades of the same color (e.g., light green and dark green) turned out to be confusing/conflicting with respect to surrounding light.

Based on our initial experimentation and testing, as described above, we finally decided to stick to a maximum of 8 human-distinguishable colors. With 8 colors, we can encode 3-bits of SAS data at a time. These eight colors are comprised of 3 primary colors (Red, Green, Blue), 3 secondary colors (Yellow, Magenta, Cyan) and two tertiary colors (Orange and Violet). With the help of some initial testing, we assured ourselves that these 8 colors are unambiguous as displayed both on a monitor screen and on a multi-color LED. We used 24-bit RGB model for the colors; thus above eight colors have the following RGB values: Red (255,0,0), Green (0,255,0), Blue (0,0,255), Yellow (255,255,0), Magenta (255,0,255), Cyan (0,255,255), Orange (255,127,0) and Violet (127,0,255). We used primary, secondary and tertiary colors for color pairing because these colors are easy to generate on a multi-color LED. The primary colors are directly generated by LED cathodes and mixing the primary colors in different ratios generated secondary and tertiary colors.

4.2 Generating “Human-Distinguishable” Colors on a Multi-Color LED

Multi-Color LED [2] is a specialized form of LED which has one common Anode and 3 Cathodes for three primary colors (Red, Green and Blue). Each of the primary colors can be produced by directly turning ON each cathode (keeping others OFF). For producing secondary colors, we need to turn ON two of the cathodes concurrently. For example, Yellow is a combination of Green and Red. So, turning ON the green and red cathodes generated Yellow. Similarly, Magenta is produced by turning ON the red and blue cathodes concurrently and cyan is produced by turning on the green and blue cathodes concurrently. Tertiary colors Orange (Red + Yellow = 2× Red + Green) and Violet (Blue + Magenta = 2× Blue + Red) are produced by turning ON two cathodes simultaneously; but, currents are varied between cathodes to produce the tertiary colors. For example, to produce Orange which is a combination of Red and Green with ratio 2:1, current flow in Red cathode is kept twice than that of Green cathode. Similarly, Violet is produced by passing twice the amount of current in Blue cathode than that of Red cathode.

For varying the current on different cathodes, we designed a circuit (as shown in Figure 1(a)), using NPN transistors, different resistors and controlled it through a computer by sending the data through the parallel port. Each cathode of the multi-color LED is controlled by 3 pins of parallel port connected with different values of resistors. By sending 9-bit binary data to parallel port, current flows in cathodes are controlled.



(a) Without color mixer coop

(b) With color mixer coop

Fig. 1. Multi-Color LED Circuit on Breadboard

The multi-color LED was enclosed with a black plastic hollow coop and covered with thin layer of tissue paper from outside (as shown in Figure 1(b)). This was done in order to generate the secondary and tertiary colors by properly mixing the primary colors.

4.3 Implementation: Transmission and Decoding

We developed an application in Visual C# to control the LED controller circuit on breadboard connected with a computer through the parallel port (as shown in Figure 1). The SAS data is mapped onto color values and color values are used to activate the corresponding cathodes of the LED in order to generate the human-distinguishable colors. When the user starts the pairing process, the application starts showing each color on the LED first and then asks the user to select the corresponding color on the screen from a list of all possible human-distinguishable colors. After selection of each color by the user, the application turns OFF the LED and asks the user to verify whether or not the LED turned OFF and accordingly press “Yes” or “No” button (respectively) to proceed and display the next color. This is done in order to keep the transmission resistant against insertion, deletion, delay and/or replay of synchronization signals between the two devices. A synchronization signal is sent, over the wireless channel, from device B with color input screen to device A displaying colors, as soon as the user selects the color on B. This instructs device A to now show the next color. Turning OFF of the LED indicates to the user that the synchronization signal has been correctly received by device A; if the LED stays ON, it indicates a synchronization error/attack.

The above process continues until the whole SAS data is transmitted encoded through human-distinguishable colors unless there is synchronization error (i.e., the LED is not turned OFF and the user presses on “NO” button or vice versa). After transmission (through human distinguishable colors on LED) and reception (from users’ color selection on screen) of the SAS data , the application shows the result (failure/success) of pairing. If there are no synchronization errors and if the SAS strings match, the pairing is deemed successful; otherwise, the pairing fails.

For 15-bit SAS data and N human-distinguishable colors, we require $\frac{15}{log_2 N}$ “passes” for transmission of SAS data. In each pass, one of N colors is shown by the multi-color LED and user selects the corresponding color from N colors on screen.

5 “Alphanumeric Pairing” Using a Sixteen-Segment Display

Sixteen Segment Display (SSD) [3] is an inexpensive, commercially available minimal alphanumeric display. It is capable of showing all the (capital) English alphabets and all digits (0-9). Due to its low cost, small size, availability and good layout, SSD is quite suitable to be incorporated for pairing operation.

In the “alphanumeric pairing” method, the SAS data is encoded into alphanumeric characters and displayed on the SSD of device A one-by-one. The user simply reads each displayed character and types it onto the keypad of device B. Similar to the pairing method based on the multi-color LED (as described in previous section), after each character is typed in by the user, the display is turned OFF and the user is asked to verify whether it is turned OFF or not, before displaying the next character. In this manner, the SAS data is transferred from device A to device B, while maintaining synchronization. Once device B decodes the received SAS data, it compares it with its own local copy; and accordingly accepts or rejects the pairing instance.

5.1 Encoding of SAS Data into Alphanumeric Characters

SSD can show 10 digits and 26 capital letters of English alphabet; i.e., a total of 36 alphanumeric characters. We wanted to keep the set of characters as unambiguous as possible so that the users with no prior knowledge of the character layout on SSD can easily identify the characters. To this end, we decided to discard 4 characters, ‘0’, ‘O’, ‘5’, and ‘S’, which appear quite ambiguous (without prior knowledge of the layout, users might mistake a ‘0’ for a ‘O’ or a ‘5’ for an ‘S’, and vice versa). This left us with a character space of size 32.

For 15-bit SAS transmission using 32 alphanumeric characters, a total of $\frac{15}{log_2(32)} = 3$ passes, i.e., 3 characters, need to be transferred between the two devices. After each

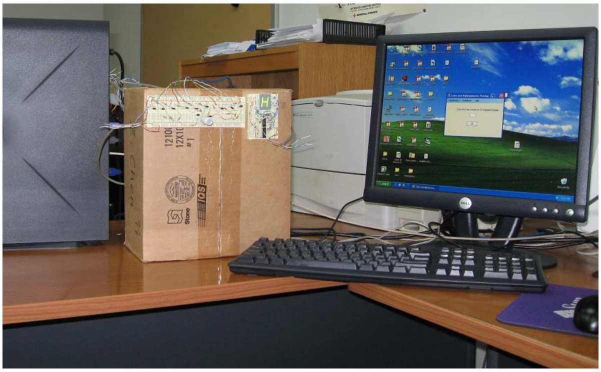


Fig. 2. Overall Experimental Setup of the Color and Alphanumeric Pairing (the cardboard box with the multi-color LED and the SSD, simulates, e.g., an access point in a cyber-cafe; the desktop simulates the laptop.)

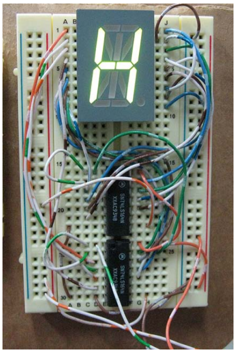


Fig. 3. Sixteen Segment Display (displaying character ‘H’)

pass, the SSD display is turned OFF and the user is asked to verify whether it is indeed OFF and accordingly press "Yes" or "No" button to proceed.

To implement our new pairing method using SSD, we extended our C# application we developed for color-based pairing (as described in previous section). The SSD was connected with the parallel port of the computer via two "serial-in-parallel-out" shift registers. Serial data and clocks were sent from the parallel port to the shift registers and shift registers supplied the data in parallel to the SSD. Figure 3 shows the snapshot of the setup of the SSD we used.

6 Experiments and Results

6.1 Experimental Setup

To test our color and alphanumeric pairing methods, we used the following set-up. The application which controls the multi-color LED and alphanumeric display is running on a DELL Desktop computer (1.8 GHz CPU, 1 GB RAM, WinXP Pro SP2) connected with multi-color LED on breadboard and sixteen segment display (as shown in Figure 2) through parallel port (DB25 Connector). This computer works as both transmitter and receiver of SAS data in both color-based and alphanumeric pairing. In color pairing, the breadboard with multi-color LED connected with parallel port simulates the transmitting device and application running on computer having interface for selecting colors simulates the other device. Similarly, the breadboard with sixteen segment display connected with parallel port of computer simulates one device and application running on the desktop computer having character input interface simulates another device in alphanumeric pairing.

For color pairing circuit, we used one multi-color LED (08L5015RGBC) [2], 12 NPN Transistors (ZTX450), 3 categories (0.5k, 1k, 2k Ohms) resistors - 3 of each value and nine 10k Ohm resistors. For alphanumeric pairing circuit, we used one sixteen segment display (AND-8010GCLB) [3], two serial-in-parallel-out shift registers (SN74LS164). Both the color and alphanumeric pairing circuits are powered from DC power source of the computer.

As mentioned in prior sections, an application running on desktop computer is developed in Microsoft Visual C# for controlling both the circuits of color and alphanumeric pairing. The application also supports all necessary functionality for an automated user testing. The application accepts the inputs from users in both color and alphanumeric pairing, shows the result of pairing and finally records the users feedback as part of the usability testing. The application also keeps track of users inputs and timings and logs the result of pairing and all necessary information regarding users background and their feedback. A couple of screen-shots of the execution of our application for both color and alphanumeric pairing are shown in Figures 4 and 5.

6.2 Usability Testing

In order to test how both of our pairing methods fare with users, and especially to figure out if the users are easily and correctly able to transfer the colors and alphanumeric characters as displayed by the multi-color LED and the SSD, respectively, we performed a thorough and systematic usability study. We focused on a common security application of detecting Evil Twin access points (as in [18], but in a rushing user resistant manner). Note that this only requires unidirectional authentication. Thus, we did not incorporate in our tests the final step of the protocol of [19] whereby the user accepts or rejects the pairing on device A based on the pairing outcome shown by device B. See Figure 2 depicting our experimental set-up.

6.3 Testing Framework

For creating a user-friendly but realistic testing framework, we extended the circuit controller application (running on the desktop computer) by implementing the usability testing and user feedback collection functionality on it.

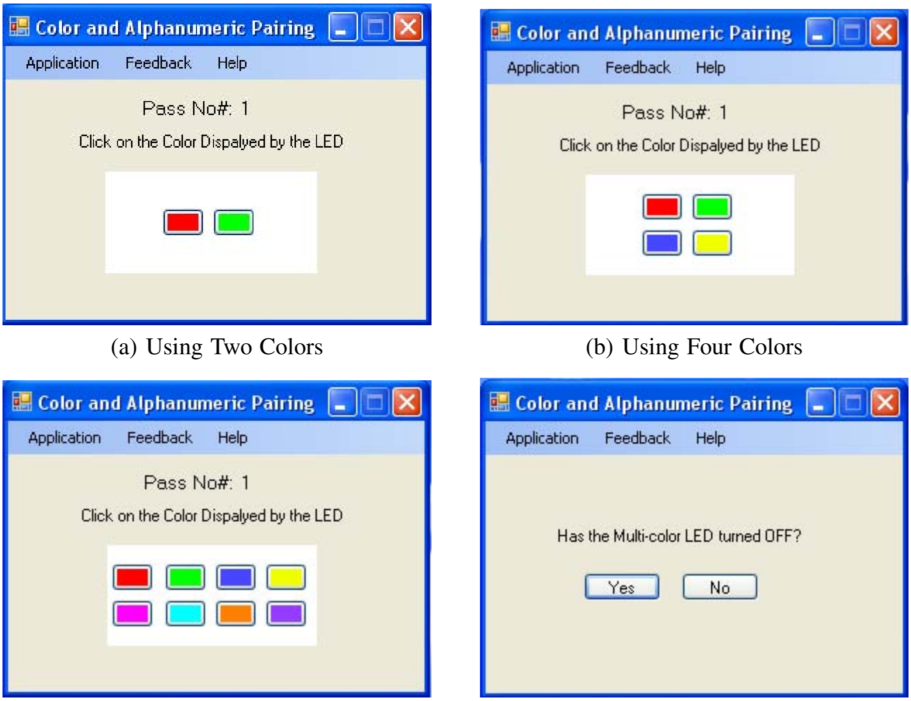
For color pairing, the users are instructed to transfer the colors as displayed by the multi-color LED to the desktop screen by clicking on the corresponding "color buttons". For testing our color pairing method with respect to pairing time and user errors, we conducted our usability testing in 3 settings - using 2, 4 and 8 colors. For 15-bit SAS string, "2-Color" pairing method requires $\frac{15}{log_2 2} = 15$ passes. Similarly, "4-Color" pairing requires $\lceil \frac{15}{log_2 4} \rceil = 8$ passes and "8-Color" pairing requires 5 passes. For all the settings, each pass is comprised of four steps: (1) Displaying of a SAS-encoded color on the LED, (2) Selection of the color by the user on desktop screen, (3) Turning off of the LED (i.e., with no color being displayed on the LED) and (4) Verification by the user whether the LED is in OFF state.

For the 2-Color pairing method, we selected first two colors (Red and Green) out of the primary colors (Red, Green and Blue). For the 4-Color pairing method, we selected primary colors (Red, Green, Blue) and first color (Yellow) out of the of secondary colors (Yellow, Magenta and Cyan). For the 8-Color pairing method, we selected primary colors (Red, Green, Blue), secondary colors (Yellow, Magenta, Cyan) and two colors (Orange and Violet) out of tertiary colors (Azure, Violet, Rose, Orange, Chartreuse and Aquamarine). A few screen-shots of user interfaces for color pairing methods are shown in Figure 4.

For testing the alphanumeric pairing method with respect to pairing time and user errors, the application is configured to take alphanumeric inputs from the users. Users were instructed to input the character, displayed on sixteen segment display, onto the desktop keyboard. For 32 alphanumeric characters and 15-bit SAS data, it requires $\frac{15}{log_2 32} = 3$ passes for the pairing process. Each pass is comprised of the following steps: (1) Displaying of a SAS-encoded character on the sixteen-segment display, (2) Inputting the corresponding character on desktop keyboard, (3) Turning off of the display (i.e., with no character being displayed on the SSD) and (4) Verification by the user whether the SSD is in OFF state. The application converted all characters to uppercase on the input text-box and displayed the result of pairing after the completion of 3 passes. A couple of screen-shots of user interfaces for the alphanumeric pairing are shown in Figure 5.

6.4 Test Cases

For each of the three methods of color pairing (i.e., using 2, 4 and 8 colors), two test cases were created; thus, a total of 6 test cases were executed by each user for color



Pass No#: 1

Click on the Color Dispalyed by the LED

(a) Using Two Colors

Pass No#: 1

Click on the Color Dispalyed by the LED

(b) Using Four Colors

Pass No#: 1

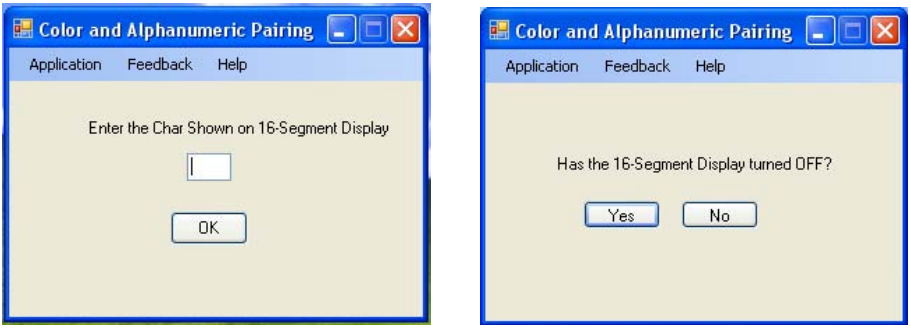
Click on the Color Dispalyed by the LED

(c) Using Eight Colors

Has the Multi-color LED turned OFF?

(d) Turn OFF Verification Screen

Fig. 4. Usability Testing of Color Pairing Using Multi-Color LED



Enter the Char Shown on 16-Segment Display

(a) Character Input Interface

Has the 16-Segment Display turned OFF?

(b) Turn OFF Display Checking Screen

Fig. 5. Usability Testing of Alphanumeric Pairing

pairing. In each color pairing method, all the colors of that particular method appeared at least once (in other words, no user missed out a single color) and the colors appeared in random order on the multi-color LED.

For alphanumeric pairing, 15-bit SAS data requires each test case to show $\frac{15}{log_2 32} = 3$ alphanumeric characters. Thus, it required $\lceil \frac{32}{3} \rceil = 11$ test cases to show all the 32 alphanumeric characters at least once to each user. Alphanumeric characters were shown randomly on the sixteen segment display and each user executed a total of 11 test cases of alphanumeric pairing.

6.5 Test Participants

We recruited 20 subjects for the usability testing of both color and alphanumeric pairing. Subjects were chosen on a first-come first-serve basis from respondents to recruiting posters and email advertisements. At the end of the tests, the participants were asked to fill out an on screen questionnaire through which we obtained user demographics and their feedback on the methods tested.

Recruited subjects were mostly university students, both graduate and undergraduate, with CS and non-CS backgrounds. This resulted in a fairly young (ages between 18-35 [*mean*=24.15, *se*=0.7549]), well-educated participant group. All participants were regular computer and cell phone users. 18 out of 20 participants reported they have previously used a wireless accessory such as access point/modem/router. 12 out of 20 participants reported they have previously used a bluetooth device such as bluetooth-headset, mouse or keyboard. Eight participants were familiar with the vulnerability of an un-encrypted wireless channel and five participants chose the statement “Un-encrypted wireless channel isn’t vulnerable to attack” and 7 participants were not sure about the statement. None of the study participants reported any physical impairments that could have interfered with their ability to complete given tasks and none of them had any visual disability, color vision problem or color blindness. The gender split was: 4 females and 16 males.

6.6 Testing Process

Our study was conducted in a graduate student laboratory of our university. Each participant was given a brief overview of our study goals and our experimental set-up. Each participating user was then asked to follow on-screen instructions on the desktop computer for both color and alphanumeric pairing. No training of any sort was given. Basically, the participants played the role of the user in the color and alphanumeric pairing process i.e., they transferred colors shown by the multi-color LED and alphanumeric characters shown by the Sixteen Segment Display to the application running on Desktop Computer. Each user completed 6 color pairing tests (two test cases for each category using 2, 4 and 8 colors) and 11 alphanumeric pairing test cases. Pairing outputs, user interactions throughout the tests and timings were logged automatically by the testing framework.

After completing the deputed test cases for both the color and alphanumeric pairing in the above manner, the participants were asked to give some qualitative feedback on the tested methods. For color pairing, participants were asked to score on a 1-10 scale (1-Low, 10-High) how distinct the colors were as shown by the monitor screen and by the multi-color LED; how easy they found to read and transfer the colors from multi-color LED to monitor screen. Users were also asked to choose- which color pairing method they preferred the most: 2-Color, 4-Color or 8-Color pairing.

For alphanumeric pairing, participants were asked to score on a 1-10 scale (1-Low, 10-High) how distinct were the alphanumeric characters as shown by the sixteen segment display, how easy they found to read and transfer the characters from sixteen segment display to the computer.

Participants' demographic information such as age, gender, educational qualification, visual and color vision disability, computer, wireless and bluetooth device usage experience and knowledge on security of wireless channel were all collected through this questionnaire. All user data and feedback were logged by the testing framework for later analysis.

6.7 Test Results

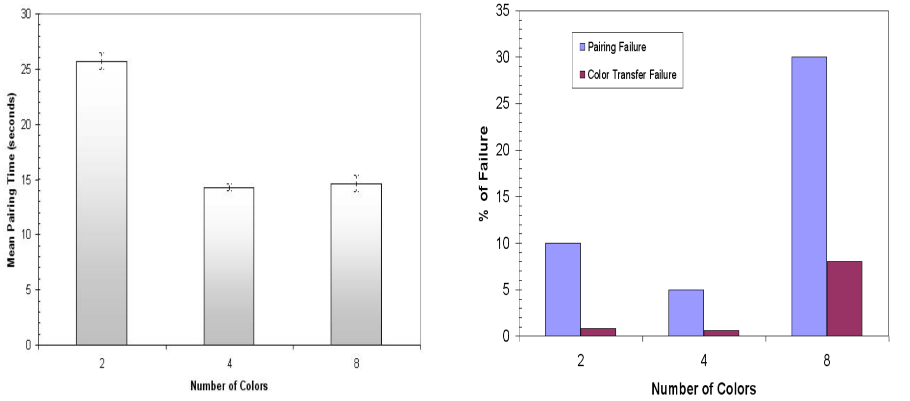
Each of our 20 subjects executed 6 color pairing and 11 alphanumeric test cases, leading to a total of 120 color pairing (i.e., 40 test cases for each of 2-Color, 4-Color, and 8-Color pairing methods) and 220 alphanumeric pairing test cases.

Errors. Most of the test cases completed successfully giving expected results. In some cases, however, we observed a few errors, which we categorize and describe below.

- *Color Pairing Errors:*
  In the 2-Color pairing method, 4 users clicked on wrong colors for a total of 5 times in 4 test cases. Thus, 4 test cases failed out of 40 test cases. So, *pairing failure* rate = $\frac{4}{40} \times 100\% = 10\%$. Users failed to transfer 5 colors out of $40 \times 15 = 600$ colors. So, *color transfer failure* rate = $\frac{5}{600} \times 100\% = 0.83\%$.
  In the 4-Color pairing method, 2 users clicked on wrong colors for a total of 2 times in 2 test cases. This led to a *pairing failure* rate = $\frac{2}{40} \times 100\% = 5\%$ and a *color transfer failure* rat e= $\frac{2}{320} \times 100\% = 0.625\%$.
  In the 8-Color pairing method, 12 users clicked on wrong color for a total of 16 times in 12 test cases, leading to a *pairing failure* rate = $\frac{12}{40} \times 100\% = 30\%$. and a *color transfer failure* rate = $\frac{16}{200} \times 100\% = 8\%$.
  The graph presented in Figure 6(b) depicts the pairing failure rates and color transfer failure rates for the three color pairing methods.
- *Alphanumeric Pairing Errors:*
  In the alphanumeric pairing, 9 users made a total of 12 errors in 12 test cases (i.e., single character errors in each failed test case) out of 220 test cases as listed in Figure 8.
  Therefore, the *pairing failure* rate = $\frac{12}{220} \times 100\% = 5.45\%$. 20 users transferred a total of $20 \times 11 \times 3 = 660$ characters and out of them, 12 characters were transferred as incorrectly So, *character transfer failure* rate = $\frac{12}{660} \times 100\% = 1.818\%$.

Test Timing. The mean pairing time of color pairing using 2, 4 and 8 colors are shown in the graph of Figure 6(a). Clearly each of the color pairing methods requires less than 30 seconds of pairing time and the 4-Color pairing method is the fastest of them all [*mean*=14.289 seconds, *se*=0.3138], whereas the 2-Color is the slowest [*mean*=25.7318 seconds, *se*=0.7545].

The average time taken by each user (over the 11 test cases) to perform the alphanumeric pairing is depicted in Figure 7. Clearly, it shows that most of the users completed the alphanumeric pairing within 15 seconds [*mean*=9.393 seconds, *se*=0.2670].



(a) Pairing Time with Standard Error

(b) Pairing and Color Transfer Failure Rates

Fig. 6. Results of Color Pairing - using 2, 4 and 8 Colors

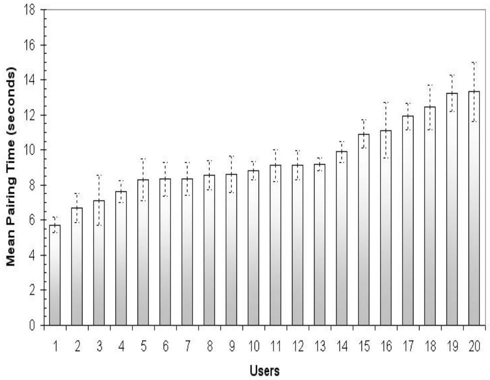


Fig. 7. Result of Alphanumeric Pairing: User Timing with Standard Error (users are sorted by average time)

Displayed on SSD	Transferred by User	# of Occurrences (out of 660)
G	6	4
Q	0	4
1	I	3
B	D	1

Fig. 8. User Errors in Reading and Transferring Alphanumeric Characters

These timings are commensurate with the timings of the pairing methods presented in [17] [18]. Recall that the latter methods are not resistant to a rushing user behavior.

User Feedback. The user feedback was collected on the distinctiveness of all 8 colors as shown by the multi-color LED and on monitor screen and on the easiness to transfer these colors. It was assumed that distinctiveness and unambiguity of these 8 colors would automatically imply the distinctiveness and unambiguity of the colors used in the 2-color and 4-color pairing methods. As our results show, most users found the methods robust and quite easy to work with. The qualitative results we obtained through the user feedback questionnaire on color pairing are shown in Table 1.

The users were also asked to compare the three methods in terms of the distinctiveness of underlying colors, ease of transfer and selection of colors and overall work-load in the whole process. In this respect, 2 out of 20, i.e., 10% users preferred the 8-color

Table 1. User Feedback Score [1 (Low) - 10 (High)] with Standard Error (*se*) on Color Pairing

Basis	**Score**
Distinctness of the colors on Monitor Screen	9.45 (*se*=0.1352)
Distinctness of the colors on Multi-Color LED	7.4 (*se*=0.3934)
Easiness of color transfer from Multi-Color LED to Monitor Screen	8.6 (*se*=0.3276)

Table 2. User Feedback Score [1 (Low) - 10 (High)] with Standard Error (*se*) on Alphanumeric Pairing

Basis	**Score**
Distinctness of the characters on 16-Segment Display	8.85 (*se*=0.2542)
Easiness of character transfer from 16-Segment Display to Computer	9.05 (*se*=0.3118)

pairing method, 10 out of 20, i.e., 50% users preferred the 4-color pairing method and 8 out of 20; i.e., 40% users preferred the 2-color pairing method.

The results of the user feedback questionnaire on alphanumeric pairing are depicted in Table 2. Similar to the color pairing methods, most users found the method robust and quite easy to work with. We did not find any notable correlation of the subjects' age, gender and technical expertise with the results obtained for both color and alphanumeric pairing.

7 Conclusions and Future Work

In this paper, we addressed a challenging problem of pairing interface-constrained devices in a rushing user resistant manner. We can draw the following conclusions from the results of usability testing of our color and alphanumeric pairing methods.

Among the color pairing methods, the 4-color method turns out to be a clear winner in terms of pairing speed, errors and usability. This is mainly due to the reason that the four colors used in the 4-color pairing method are quite distinct and unambiguously identifiable by human users, as opposed to the eight colors used in the 8-color pairing method. In comparison to the 2-color method, on the other hand, the 4-color method requires fewer passes, thus speeding up the pairing process and reducing user burden.

As our results show, the alphanumeric pairing method also turned out to be quite robust to errors, fast and user-friendly. The errors resulting through this method could potentially be further reduced by using less confusing non-alphabetic characters.

Both 4-color pairing and alphanumeric pairing methods can also be easily adopted on devices which have good quality output interfaces (such as a full display on a cell phone). Notice that on such devices, there will be no need for synchronization between the two devices, as all of SAS data (e.g., all colors and all digits) can be displayed on one single screen. In fact, the "Copy-and-Confirm" method of [25] is nothing but our alphanumeric pairing method for devices with good quality displays.

Between the 4-color pairing and alphanumeric pairing methods, we find the latter to be slightly faster, but, the former to be slightly more robust to errors. In terms of usability also, the two methods turned out to be comparable. However, since a multicolor LED is smaller in size and slightly cheaper than the sixteen segment display, we believe that 4-color pairing would be a more practical choice. Moreover, LED colors can also be comprehended from a distance (e.g., in case of a wall-mounted access point). To support the 4-color pairing method, device manufacturers would only need to use a multi-color LED in place of regular LED(s) which are quite common on most devices.

In conclusion, our results show that the 4-color pairing method is quite appropriate for most pairing scenarios: it is fast, robust, user-friendly, resistant against rushing user behavior, and can be incorporated on most devices with only a little modification. Based on our testing, 4-color pairing method turns our to be ideal for defending against a common threat of evil twin access points, in a rushing user resistant manner. Alphanumeric pairing is suitable for devices that can afford to have a sixteen segment display.

In our current design of the 4-color pairing method, we did not consider “color-blindness”, a common visual disability. It is found that most color-blind people are “red-green” color blind (i.e., they can not distinguish between red and green colors) [1]. To address color-blindness, one could select distinct colors other than red and green in our 4-Color pairing method. In our future work, we will design and evaluate such a variation of the 4-color pairing method.

Acknowledgments. The authors would like to thank Vikram Padman and Kurt Rosenfeld for their help and discussion on the setup involving Multi-Color LED.

References

1. Color blindness. On-line article Published by University of Illinois Eye and Ear Infirmary, `http://www.uic.edu/com/eye/LearningAboutVision/EyeFacts/ColorBlindness.shtml`
2. Datasheet and Specification for Multi-Color LED. Electronix Express/RSR Electronics, `http://www.elexp.com/a_data/0815015rgbc.pdf`
3. Datasheet and Specification of Sixteen Segment Display, `http://www.purdyelectronics.com/pdf/AND8010-B.pdf`
4. Balfanz, D., Smetters, D., Stewart, P., Wong, H.C.: Talking to strangers: Authentication in ad-hoc wireless networks. In: Network & Distributed System Security (NDSS) (2002)
5. 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, p. 453. Springer, Heidelberg (2001)
6. Dhamija, R., Tygar, J.D., Hearst, M.A.: Why phishing works. In: International Conference for Human-Computer Interaction (CHI) (2006)
7. Gehrmann, C., Mitchell, C.J., Nyberg, K.: Manual authentication for wireless devices. RSA CryptoBytes 7(1), 29–37 (Spring 2004)
8. Glasbey, C., van der Heijden, G., Toh, V., Gray, A.: Colour displays for categorical images. Color Research and Application 32, 304–309 (2007)
9. Goldberg, I.: Visual Key Fingerprint Code (1996), `http://www.cs.berkeley.edu/iang/visprint.c`
10. Goodrich, M.T., Sirivianos, M., Solis, J., Tsudik, G., Uzun, E.: Loud and Clear: Human-Verifiable Authentication Based on Audio. In: International Conference on Distributed Computing Systems (ICDCS) (2006)

11. Kuo, C., Walker, J., Perrig, A.: Low-cost manufacturing, usability, and security: An analysis of bluetooth simple pairing and wi-fi protected setup. In: Dietrich, S., Dhamija, R. (eds.) USEC 2007. LNCS, vol. 4886, pp. 325–340. Springer, Heidelberg (2007)
12. Laur, S., Asokan, N., Nyberg, K.: Efficient mutual data authentication using manually authenticated strings. In: Pointcheval, D., Mu, Y., Chen, K. (eds.) CANS 2006. LNCS, vol. 4301, pp. 90–107. Springer, Heidelberg (2006)
13. McCune, J.M., Perrig, A., Reiter, M.K.: Seeing-is-believing: Using camera phones for human-verifiable authentication. In: IEEE Symposium on Security and Privacy (2005)
14. Pasini, S., Vaudenay, S.: An optimal non-interactive message authentication protocol. In: The Cryptographers' Track at the RSA Conference (CT-RSA) (2006)
15. Pasini, S., Vaudenay, S.: SAS-Based Authenticated Key Agreement. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 395–409. Springer, Heidelberg (2006)
16. Perrig, A., Song, D.: Hash visualization: a new technique to improve real-world security. In: International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC) (1999)
17. Prasad, R., Saxena, N.: Efficient Device Pairing using Human-Comparable Synchronized Audio Visual Patterns. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 328–345. Springer, Heidelberg (2008)
18. Roth, V., Polak, W., Rieffel, E., Turner, T.: Simple and effective defenses against evil twin access points. In: ACM Conference on Wireless Network Security (WiSec) (2008)
19. Saxena, N., Ekberg, J.-E., Kostiainen, K., Asokan, N.: Secure device pairing based on a visual channel. In: IEEE Symposium on Security & Privacy, short paper (2006)
20. Saxena, N., Uddin, M.B., Voris, J.: Universal Device Pairing using an Auxiliary Device. In: Symposium On Usable Privacy and Security (SOUPS) (2008)
21. Soriente, C., Tsudik, G., Uzun, E.: BEDA: Button-Enabled Device Association. In: International Workshop on Security for Spontaneous Interaction (IWSSI) (2007)
22. Soriente, C., Tsudik, G., Uzun, E.: HAPADEP: Human Asisted Pure Audio Device Pairing. In: International Information Security Conference (ISC), Taipei, Taiwan (September 2008)
23. Stajano, F., Anderson, R.J.: The resurrecting duckling: Security issues for ad-hoc wireless networks. In: Security Protocols Workshop (1999)
24. Suomalainen, J., Valkonen, J., Asokan, N.: Security associations in personal networks: A comparative analysis. In: Stajano, F., Meadows, C., Capkun, S., Moore, T. (eds.) ESAS 2007. LNCS, vol. 4572, pp. 43–57. Springer, Heidelberg (2007)
25. Uzun, E., Karvonen, K., Asokan, N.: Usability analysis of secure pairing methods. In: Dietrich, S., Dhamija, R. (eds.) USEC 2007. LNCS, vol. 4886, pp. 307–324. Springer, Heidelberg (2007)
26. Vaudenay, S.: Secure communications over insecure channels based on short authenticated strings. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 309–326. Springer, Heidelberg (2005)
27. Čagalj, M., Čapkun, S., Hubaux, J.-P.: Key agreement in peer-to-peer wireless networks. Proceedings of the IEEE 94(2), 467–478 (2006)

How to Extract and Expand Randomness: A Summary and Explanation of Existing Results*

Yvonne Cliff, Colin Boyd, and Juan Gonzalez Nieto

Information Security Institute, Queensland University of Technology
GPO Box 2434, Brisbane Qld 4001, Australia
y.cliff@isi.qut.edu.au, {c.boyd,j.gonzaleznieto}@qut.edu.au

Abstract. We examine the use of randomness extraction and expansion in key agreement (KA) protocols to generate uniformly random keys in the standard model. Although existing works provide the basic theorems necessary, they lack details or examples of appropriate cryptographic primitives and/or parameter sizes. This has lead to the large amount of min-entropy needed in the (non-uniform) shared secret being overlooked in proposals and efficiency comparisons of KA protocols. We therefore summarize existing work in the area and examine the security levels achieved with the use of various extractors and expanders for particular parameter sizes. The tables presented herein show that the shared secret needs a min-entropy of at least 292 bits (and even more with more realistic assumptions) to achieve an overall security level of 80 bits using the extractors and expanders we consider. The tables may be used to find the min-entropy required for various security levels and assumptions. We also find that when using the short exponent theorems of Gennaro et al., the short exponents may need to be much longer than they suggested.

Keywords: randomness extraction, randomness expansion, key agreement, key exchange protocols, pseudorandom function (PRF), universal hash function, leftover hash lemma (LHL).

1 Introduction

In this paper we examine the techniques available for extracting and expanding randomness in the context of key agreement (KA) protocols. In such protocols, an agreed secret key is often a random member of a given group, and not a string of bits distributed uniformly at random. However, when the key is used, e.g. as the key of a symmetric encryption scheme, it is likely that a key consisting of bits distributed uniformly at random will be necessary, requiring the use of randomness extraction, and possibly randomness expansion techniques.

[*] This is an extended abstract. The full version is available at http://eprint.iacr.org/2009/136. Research funded by Australian Research Council through Discovery Project DP0666065.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 53–70, 2009.

Informally, a randomness extractor is a family of functions keyed by a random but public value, where the input to each function is a value with high entropy, and the output is indistinguishable from a uniformly random bit string. Unfortunately, the number of bits of entropy in the input must usually be much larger than the number of bits in the output for practical security parameters.

A randomness expander, or pseudo-random function family (PRFF), is a family of functions keyed by secret, uniformly random strings, with each function taking as input any publicly known value and outputting a value indistinguishable from one distributed uniformly at random.

When only one relatively short uniformly random key is required of a KA protocol, the output of a randomness extractor may be used as the required key. However, it is more likely that the output of the extractor will be used to key a randomness expander, to provide a longer key or multiple keys, e.g. when one random group member is used to derive a MAC (message authentication code) key for use in the KA protocol, as well as the final agreed secret key.

This step of converting a randomly chosen group member to a uniformly random string or strings of bits is often not discussed in papers proposing KA protocols. However, if the key derivation function is not modelled with the random oracle model, this step has a significant impact on how large the security parameter of the KA protocol needs to be to achieve proven security of a given level. As noted by Gennaro et al. [1, p.4] and Chevassut et al. [2, p.2], this point is often overlooked, particularly in protocol efficiency comparisons.

One reason randomness extraction and expansion and their effect on security parameter sizes is often overlooked may be the plethora of existing works that must be examined to obtain the necessary background knowledge, and the dearth numerical examples. Therefore, this paper provides:

- a summary of existing results on randomness extraction and expansion, including relevant definitions and theorems, and numerical examples,
- details of the short exponent discrete-log (DLSE) assumption and its use with randomness extraction and expansion (including numerical examples),
- an analysis of why assumptions made by Dodis et al. [3] in some justifications of the use of HMAC and cascade chaining (such as SHA) as randomness extractors are not realistic,
- a valuable resource for protocol designers and implementors to enable them to use security parameters of an appropriate size in efficiency comparisons and implementations, without having to examine all of the existing works,
- the observation, through the use of numerical examples for values of practical interest, that some of the theoretical results available are of limited practical value, due to the non-existence of underlying functions of an appropriate size or the availability of better methods,
- results for the standard model only; although use of a random oracle as a randomness extractor would mean that shorter parameters would be required in a protocol to achieve the same security level, making it more efficient, our aim is to describe solutions available for the standard model.

We will begin by examining the suitability of various candidates as randomness expanders, which will tell us how large a key needs to be provided by the randomness extractor. We will then examine randomness extractors, and the amount of entropy required for their input in order to extract a long enough key for the randomness expander.

Prior work includes that of Dodis et al. [3], the first to attempt to justify the use of CBC-MAC, cascade chaining and HMAC as randomness extractors in the standard model, and that of Gennaro et al. [1] who examined the use of universal hash functions as randomness extractors in conjunction with the DDH (decisional Diffie-Hellman) assumption and short exponents. Chevassut et al. [2] made some brief but interesting observations on randomness extraction and expansion in general, before providing methods of randomness extraction which are more efficient than those studied here, but are only applicable for groups of points over an elliptic curve (EC), and the group of prime order q in $\mathbb{Z}_p^*$ where $p = 2q + 1$ and is prime. Their method for EC groups requires computations in the KA protocol to be carried out on an EC as well as its twist, instead of just on the curve, and so increases the number of computations required. However, the method may be advantageous as these computations on the EC and its twist will be in smaller groups than those necessary when using the methods studied in this paper in conjunction with computations on the EC only. Fouque et al. [4] showed that the lower order bits of a member of a subgroup of $\mathbb{Z}_p^*$ may be considered random in the right circumstances. Another work of Fouque et al. [5] examined the use of HMAC as a randomness extractor when the randomness is extracted from the HMAC key, and included an analysis of the cascade construction as a randomness extractor.

2 Notation and Basic Definitions

The notation mostly follows Dodis et al. [3] and Gennaro et al. [1]. For a probability distribution $\mathcal{X}$ over a set A, the notation $x \in_{\mathcal{X}} A$ indicates that x is chosen from A according to the distribution $\mathcal{X}$. The notation $x \in_{\mathrm{R}} A$ indicates that x is chosen from A according to the uniform distribution. $\Pr_{\mathcal{X}}[x]$ indicates the probability that distribution $\mathcal{X}$ assigns to the value $x \in A$. In some cases, definitions taken from other works have been modified to make the notation consistent.

This paper uses a concrete security approach, to allow determination of the size of the parameters needed in a protocol to achieve a given level of security. Following Gennaro et al. [1], we speak of circuits of size S having a certain probability, ϵ of solving a particular problem. One may also think of a circuit of size S as a programme running in time t, where 'time' actually includes the length of the description of the programme (to avoid trivializing hard problems through the use of large precomputed tables), as well as the actual execution time of that programme [6].

We now introduce computational indistinguishability, a refinement of the notion of statistical distance (or variation distance) from probability theory. If two distributions are statistically close, they are computationally indistinguishable, although the converse is not true [7, Sect. 3.2.2].

Definition 1 ((S,ϵ)-indistinguishability [1, p.19]). *Let $\mathcal{X},\mathcal{Y}$ be two probability distributions over A. Given a circuit D (called the distinguisher) consider the following quantities:*

$$\delta_{D,\mathcal{X}} = \Pr_{x\in\mathcal{X}}[D(x)=1] \quad \textit{and} \quad \delta_{D,\mathcal{Y}} = \Pr_{y\in\mathcal{Y}}[D(y)=1] \tag{1}$$

We say that the probability distributions $\mathcal{X}$ and $\mathcal{Y}$ are (S,ϵ)-indistinguishable if for every circuit D of size $\leq S$ we have that $|\delta_{D,\mathcal{X}} - \delta_{D,\mathcal{Y}}| \leq \epsilon$.

Definition 2 (Statistical Distance [8, p.131]). *The statistical distance between two probability distributions $\mathcal{X}$ and $\mathcal{Y}$ over a set A is defined to be*[1] $\Delta[\mathcal{X};\mathcal{Y}] = \frac{1}{2}\sum_{x\in A}|\Pr_{\mathcal{X}}[x] - \Pr_{\mathcal{Y}}[x]|$.

Lemma 1 ([3, p.500]). *If two distributions have statistical distance of (at most) ϵ, they are ϵ-close. Distributions that are ϵ-close cannot be distinguished with probability better than ϵ even by a computationally unbounded adversary.*

The following lemma has a proof [1] based on the triangle inequality or "hybrid argument."

Lemma 2 ([1, p.19]). *Let three probability distributions $\mathcal{X},\mathcal{Y},\mathcal{Z}$ over a set A be such that (i) $\mathcal{X}$ is (S_1,ϵ_1) indistinguishable from $\mathcal{Y}$ and (ii) $\mathcal{Y}$ is (S_2,ϵ_2) indistinguishable from $\mathcal{Z}$. Then $\mathcal{X}$ is (S,ϵ) indistinguishable from $\mathcal{Z}$ where $S = \min(S_1,S_2)$ and $\epsilon = \epsilon_1+\epsilon_2$.*

We now focus on describing how much randomness is in a probability distribution, defining min-entropy and its computational analogue.

Definition 3 (Min-entropy [1, p.9]). *If $\mathcal{X}$ is a probability distribution over A, the min-entropy of $\mathcal{X}$ is $\text{min-ent}(\mathcal{X}) = \min_{x\in A:\Pr_{\mathcal{X}}[x]\neq 0}(-log_2(\Pr_{\mathcal{X}}[x]))$. (Note that if $\mathcal{X}$ has min-entropy t then for all $x\in A$, $\Pr_{\mathcal{X}}[x]\leq 2^{-t}$.)*

Definition 4 (Computational entropy t [1, p.10]). *A probability distribution $\mathcal{Y}$ has (S,ϵ) computational entropy t if there exists a probability distribution $\mathcal{X}$ that is (S,ϵ) indistinguishable from $\mathcal{Y}$ and min-ent$(\mathcal{X})\geq t$.*

Definition 5 (Function Family [6, adapted from full paper p.7]). *A function family $f : K\times D\rightarrow R$ (also denoted $\{f_\kappa\}_{\kappa\in K}$), where K is a non-empty set of keys, is a collection of functions, $f_\kappa(\cdot)\overset{\text{def}}{=} f(\kappa,\cdot)$ for $\kappa\in K$, from a domain, D, to a range, R. We call f a permutation family if $D=R$, and for each key $\kappa\in K$, f_κ is a permutation on D.*

Definition 6 (Truly Random Function (TRF) [5,6]). *Denote the set of all functions from M to $\{0,1\}^L$ with $\text{Rand}^{M\rightarrow 2^L}$ (there are $2^{L|M|}$ such functions). A function chosen at random from $\text{Rand}^{M\rightarrow 2^L}$ is a truly random function (TRF) with input domain M and output domain $\{0,1\}^L$.*

[1] Gennaro et al.'s definition [1] is twice this value, but seems erroneous when compared with others [8,3,7] .

A TRF may be implemented by an oracle that, for each new oracle query, generates an output selected at random from $\{0,1\}^L$, and for oracle queries that are not new, replies with the same output as previously given for that input.

Definition 7 (Cascade Construction [5]). *The cascade construction (also known as keyed Merkle-Damgard cascade chaining) is the construction used for iterated hash functions. Let* $H : \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^c$ *denote an iterated hash function, and let* $h : \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$ *(the so-called compression function) be a family with key space* $\{0,1\}^c$*. The cascade construction of* h *is the function* $h^* : \{0,1\}^c \times \left(\{0,1\}^b\right)^* \to \{0,1\}^c$ *defined by:*

$$y_0 = a, y_i = h(y_{i-1}, x_i) \textit{ and } h^*(a, x) = y_n$$

where $x = (x_1, \ldots, x_n)$ *is a* $n \cdot b$ *bit string and* $a \in \{0,1\}^c$*. To construct* H*, messages must be padded to an exact multiple of* b *bits. The padding, denoted* $\mathrm{pad}(|x|)$*, is a function of the input length,* $|x|$*. Let* $x_{pad} = x \parallel \mathrm{pad}(|x|)$*. Then* H *is defined by* $H(a, x) = h^*(a, x_{pad})$*.*

Let $1 \le c' \le c$ *be an integer and let* $\mathrm{msb}_{c'}(\cdot)$ *denote the* c' *most significant bits of a bit string. For any function* H *with range* $\{0,1\}^c$*, we define for every input* x *the truncated iterated hash function* $\tilde{H}(x) = \mathrm{msb}_{c'}(H(x))$*; e.g. SHA-384 has* $c' = 384$ *and* $c = 512$*.*

Definition 8 (NMAC [5]). $\mathrm{Nmac} : \{0,1\}^c \times \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^{c'}$ *is a hash function family constructed from a (possibly truncated) iterated hash function* $\mathrm{Hash} : \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^{c'}$*. If* $(k_1, k_2) \in (\{0,1\}^c)^2$ *is a couple of keys and* $x \in \{0,1\}^*$ *is the input, the definition of NMAC is* $\mathrm{Nmac}^{\mathrm{Hash}}(k_1, k_2, x) = \mathrm{Hash}(k_2, \mathrm{Hash}(k_1, x))$*.*

Definition 9 (HMAC [5]). *HMAC is a hash function from* $\{0,1\}^* \times \{0,1\}^*$ *to* $\{0,1\}^{c'}$*. Let ipad and opad be two b-bit strings and IV be a c-bit string. Let Hash* $: \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^{c'}$ *be the (possibly truncated) iterated hash function with compression function* $h : \{0,1\}^c \times \{0,1\}^b \to \{0,1\}^c$*. If the key* k *is a bit string from* $\{0,1\}^b$*, then*

$$\begin{aligned}\mathrm{Hmac}_{IV}^{\mathrm{Hash}}(ipad, opad; k, x) &= \mathrm{Hash}\left(IV, [k \oplus opad] \parallel \mathrm{Hash}\left(IV, [k \oplus ipad] \parallel x\right)\right)\\ &= \mathrm{Nmac}^{\mathrm{Hash}}(h(IV, k \oplus ipad), h(IV, k \oplus opad), x).\end{aligned}$$

If the key k *is smaller than* b *bits, then it is first padded with '0' bits to form a* b*-bit string, and this string is used as the key. If the key* k *is larger than* b *bits, it is first hashed using* Hash *to obtain a* c'*-bit digest, then padded with* $b - c'$ *'0' bits to obtain a* b*-bit string, which is then used as the key.*

3 Randomness Expansion

To ascertain the minimum output length required from the randomness extractor used, we begin by examining the randomness expander—also known as a pseudorandom function (PRF) family, or PRFF—to be used, since the output of the randomness extractor will be used as the key to the PRFF.

Definition 10 (Pseudorandom Function Family [9,6]). *A function family $f = \{f_\kappa\}_{\kappa \in K}$ is a (S, q, ϵ) pseudorandom function family (PRFF) if a circuit, $\mathcal{A}$, of size S which is given oracle access to either f_κ for $\kappa \in_R K$ or a TRF with the same domain and range as the functions in f, and makes at most q queries to this oracle, has advantage at most ϵ in distinguishing whether it has access to a random member of f or a TRF; i.e.:*

$$\epsilon \geq \mathbf{Adv}_f^{\mathrm{prf}}(q, S) \stackrel{\mathrm{def}}{=} \max_{\mathcal{A}} \left\{ \mathbf{Adv}_f^{\mathrm{prf}}(\mathcal{A}) \right\} \tag{2}$$

$$\mathbf{Adv}_f^{\mathrm{prf}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \left| \Pr[\mathcal{A}^{\mathcal{O}(\cdot)} = 1 | \mathcal{O}(\cdot) \in_{\mathrm{R}} f] - \Pr[\mathcal{A}^{\mathcal{O}(\cdot)} = 1 | \mathcal{O}(\cdot) \in_{\mathrm{R}} \mathrm{Rand}] \right| \tag{3}$$

The values $\mathbf{Adv}_f^{\mathrm{prp}}(q, S)$ and $\mathbf{Adv}_f^{\mathrm{prp}}(\mathcal{A})$, may be defined similarly for an adversary $\mathcal{A}$ against a pseudorandom permutation family, except that $\mathcal{A}$ attempts to tell the difference between the permutation family and a truly random permutation, rather than a TRF.

When one PRFF is used with various different keys (e.g. each party from a number of parties may use its own key to produce pseudorandom values from the PRFF), there is a linear decrease in security. Furthermore, the key to a PRFF may be only computationally indistinguishable from random, in which case the level of security of the PRFF and the level computational indistinguishability must be combined. This is formally stated and proven in the full version.

Function families widely believed to be pseudorandom include CBC-MAC used in conjunction with a block cipher, HMAC or the HMAC variant NMAC, and cryptographic hash functions such as SHA-1 or SHA-256 based on the cascade construction, but with the fixed IV (initialization vector) replaced with a random key. The full paper discusses the merits of each of these options in turn. Here we overview the security levels provided by each option. Some assumptions (described in the full paper) must be made on the security level of the underlying block ciphers or compression functions to arrive at the below concrete security levels.

3.1 CBC-MAC

Bellare et al. [6] have proved that CBC-MAC is a secure PRFF if the underlying block cipher is a secure pseudorandom permutation family and the input length is constant. The level of security provided depends on the block length, number of queries, q, and number of blocks of input, l. When ql is small (e.g. 2), the security level is about $k = b - 3$ bits. Otherwise, if we have $ql \leq 2^k$ (which we are assuming when we consider a security level of 2^k sufficient), then we will require $k \leq (b-2)/2$. If the block cipher to be used with CBC-MAC is AES-128, AES-192, or AES-256, then the block length, b, will be 128 bits for each of these ciphers [10]. Therefore, the level of security provided by CBC-MAC when used in conjunction with any of these ciphers will be no greater than 125 bits, and will be less for values of q and l larger than 1. Hence, CBC-MAC is likely to be an acceptable choice of randomness expander for security levels of 80 bits if the number of queries to randomness expander with a single key is small and the length of each query is also small, but inadequate for security levels of 128 bits and higher. If an unlimited number of queries or queries with a very large length are able to be made by the adversary to the randomness expander with a single key, the security level will only be $(b-2)/2 = 63$ bits when $b = 128$.

3.2 HMAC

Bellare [11] has proven that HMAC is a secure pseudorandom function if the compression function of the underlying hash function is a pseudorandom function. The analysis assumes that the key provided to HMAC is the same length as a block for the underlying hash function (i.e. b bits). To achieve a shorter key of only $2c$ bits (where c is the length of the output of the compression function), NMAC may be used, which is similar to HMAC but differs in its use of keying material. However, NMAC is generally used for analysis of HMAC only, so availability of an existing implementation is unlikely. Any implementation of NMAC will require access to the compression function underlying the hash function to be used, which may be difficult to acquire.

Hash functions likely to be used with HMAC include MD5 [12], RIPEMD-160 [13], SHA-1, SHA-256, SHA-384 and SHA-512 [14]. Table 1 shows the block size (b), compression function key and output length (c), hash function output length (c') and HMAC security level for each of these algorithms, where q is the number of queries using the same key and l is the number of blocks per query. The traditional security level is $c/2$ bits, due to the birthday based forgery attacks against iterated MACs [15] that require $2^{c/2}$ oracle queries.

3.3 Cascade Construction

Bellare, Canetti and Krawczyk [9] have provided a proof of pseudorandom function family security for cryptographic hash functions such as SHA-1 or SHA-256 based on the cascade construction, but with the fixed IV (initialization vector) replaced with a random key, provided the input is prefix-free and the underlying compression function used by the hash function is a pseudorandom function family. (It is possible to remove the prefix-free requirement by using extra keying

Table 1. Block and key size, output length, and hash and HMAC security level

Algorithm	b	c'	c	Security level $(q, l \leq 2)$		Security level (q is large)		
				for Hash	for HMAC	max. for	conservative	HMAC
				$(c-2)$	$(c-4)$	Hash $(\frac{c-20}{2})$	Hash $(\frac{c-40}{3})$	$(\frac{c-2}{2})$
MD5	512	128	128	126	124	54	29	63
RIPEMD-160	512	160	160	158	156	70	40	79
SHA-1	512	160	160	158	156	70	40	79
SHA-224	512	256	224	254	252	118	72	127
SHA-256	512	256	256	254	252	118	72	127
SHA-384	1024	512	384	510	508	246	157	255
SHA-512	1024	512	512	510	508	246	157	255

Table 2. Summary of required key lengths for a given security level when q large

Security	Key Length				
level	CBC-	Casc.	Casc.	NMAC	HMAC
(bits)	MAC	min.	consrv.		
29			128		
40			160		
54		128			
63	128			256	512
70		160			
72			256		

Security	Key Length			
level	Casc.	Casc.	NMAC	HMAC
(bits)	min.	consrv.		
79			320	512
118	256			
127			512	512
157		512		
246	512			
255			1024	1024

material, but it is unlikely to be necessary in our setting. Belare et al. provided another construction to improve security using randomization, but if the extra randomness is counted as part of the key, more keying material than HMAC is required for a similar security level.)

Table 1 shows the security level of the cascade construction using the same notation as for HMAC. Assumptions made to obtain the security levels are described in the full paper. The difference between the maximum and conservative security levels for large q is due to different assumptions concerning the efficiency of the best attack against the underlying compression function.

3.4 Key Length Summary

In summary, when $q \leq 2$, a minimum of 128 bits will be needed to key the randomness expander, e.g. using CBC-MAC or the cascade construction, achieving a security level around 125 bits. In this case, the cascade construction allows the use of a key about two bits longer than the required security level, and requires fewer key bits than using NMAC or HMAC for the same security level.

When there is no restriction on q, the cascade construction provides the lowest key length for a given security level when we take the security level as being $\frac{c-20}{2}$. However, if the more conservative security level of $\frac{c-40}{3}$ bits is used, then NMAC may be better, depending on the level of security required. Table 2 summarizes the results.

4 Randomness Extraction

Let us consider a KA protocol that allows the participating parties to agree upon a secret value, called the pre-secret, that an adversary cannot distinguish from a value drawn uniformly at random from a particular distribution, e.g. from a group in which the DDH (Decisional Diffie-Hellman) assumption holds. Furthermore, assume a randomness extractor and expander are used to derive a final key from the pre-secret, such that the final key is indistinguishable from a uniformly random bit string. As will be seen in this section, when using the techniques of randomness extraction and expansion considered in this paper, the entropy of the pre-secret must be much larger than the security level required of the final key. Therefore, if the pre-secret is from a suitable group, it may seem desirable to use the discrete-log short-exponent (DLSE) assumption to enable calculations required by the KA protocol to be more efficient, by using exponents shorter than the group order. In addition, if the KA protocol is Diffie-Hellman (DH) based, it may be desirable to use the t-DDH assumption (a relaxation of the DDH assumption) to allow the use of groups with non-prime order with the protocol. These assumptions and theorems are therefore provided in the full paper. Note that two theorems of Gennaro et al. [1] regarding use of the DLSE assumption are incorrect in their original paper and have been corrected in the full paper according to details supplied by Gennaro in a personal communication.

The most common existing randomness extractor definition is of a strong randomness extractor:

Definition 11 (Strong randomness extractor [16]). *A family of efficiently computable hash functions $H = \{h_\kappa : \{0,1\}^n \to \{0,1\}^c | \kappa \in \{0,1\}^d\}$ is called a (t, ϵ) strong randomness extractor, if for any random variable X over $\{0,1\}^n$ that has min-entropy at least t, if κ is chosen uniformly at random from $\{0,1\}^d$ and R is chosen uniformly at random from $\{0,1\}^c$, the following two distributions are within statistical distance ϵ from each other: $(\kappa, h_\kappa(X)) \cong_\epsilon (\kappa, R)$.*

By Lemma 1, the above distributions are also computationally indistinguishable from each other. Notice that the definition means that the key to the randomness extractor, κ, may be made public, yet the output of the randomness extractor, given a secret input with sufficient min-entropy, is indistinguishable from a string of bits distributed uniformly at random.

Since it is likely that X will only have computational entropy (not min-entropy) of a certain level, we introduce the following definition (which is similar to a recent definition of Fouque et al. [5] in an independent work).

Definition 12 (Strong computational randomness extractor). *A family of efficiently computable functions $H = \{h_\kappa : A \to \{0,1\}^c | \kappa \in \{0,1\}^d\}$ is a $(t, S, \epsilon, S', \epsilon')$ strong computational randomness extractor if given any probability distribution $\mathcal{X}$ over A such that $\mathcal{X}$ has (S, ϵ) computational entropy at least t, the following two probability distributions are (S', ϵ')-indistinguishable:*

$$\mathcal{H} = \{(\kappa, h_\kappa(x)) \textit{ for } \kappa \in_{\mathrm{R}} \{0,1\}^d \textit{ and } x \in_{\mathcal{X}} A\} \tag{4}$$

$$\mathcal{R}^{\mathrm{h}} = \{(\kappa, r) \textit{ for } \kappa \in_{\mathrm{R}} \{0,1\}^d \textit{ and } r \in_{\mathrm{R}} \{0,1\}^c\} \tag{5}$$

It is possible to show that a strong randomness extractor is also a strong computational randomness extractor (see the full paper). However, the converse is not necessarily true.

The above definitions assume that the key to the randomness extractor, κ, is generated afresh for each use of the randomness extractor. This may be appropriate in some protocols, where parties may have exchanged nonces with each other and can use these values to generate the key. However, it is imperative that any such nonces be authenticated (i.e. unable to be influenced by the adversary) and not subject to replay by the adversary. Otherwise, a key derived from these nonces may not be distributed uniformly at random over $\{0,1\}^d$ as required for these extractors.

When parties are unable to generate a new key, κ, each time they use a randomness extractor, the key κ may be fixed as part of the system parameters. However, this requires multiple uses of the randomness extractor with the one key. It turns out that the security of the randomness extractor decreases linearly with the number of queries to it using the same key. Protocols using this approach may be proven secure in one of two ways. As part of the proof of security of the protocol, one often focuses on the security of one particular session chosen at random from all sessions. In the proof, it may be possible to use the above definitions to prove the security of the protocol. The total number of sessions will appear as a factor in the security reduction (due to focusing on one session chosen at random from all sessions), and this will cater for the reduction in security due to multiple uses of the extractor with only one key. The other way to justify the use of a single key to the randomness extractor is via the theorems given in the full paper.

4.1 Combining Extraction and Expansion

To ascertain the security of the overall key derivation function consisting of randomness extraction and expansion, all of the relevant theorems from the full paper must be combined (e.g. extractor reuse, Diffie-Hellman assumption, short exponent theorems, expander reuse etc.). The full paper provides an example combination of theorems which is summarized here.

Let $H = \{h_\kappa : \{0,1\}^n \to \{0,1\}^c | \kappa \in \{0,1\}^d\}$ be a $(t, 2^{-e})$ strong randomness extractor, with a maximum of q_1 queries per (publicly known) randomness extractor key κ, and let $f = \{f_\lambda\}_{\lambda \in K}$ be a (S_5, q_2, ϵ_5) PRFF, with a maximum of q_2 queries per (secret) key λ. Suppose a security level of k bits is desired for the final key(s) output by f. Let G be a cyclic group of order m generated by g, such that m is odd, or $m/2$ is odd. We assume there are q_1 publicly known pairs g^{a_i}, g^{b_i} for $1 \leq i \leq q_1$, and that the $g^{a_i b_i}$ are the inputs to $h_\kappa(\cdot)$.

We consider two cases. For the first, we require $q_1 - 1$ outputs of H to be indistinguishable from random, use the other output of H to key f, and require the q_2 outputs of f using this key to be indistinguishable from random. The indistinguishable distributions are labelled $\mathcal{EEDH}$ and $\mathcal{EER}$.

In the second case, all q_1 outputs of H are used to key f, giving a total of $q_1 q_2$ outputs of f, and all of these outputs must be indistinguishable from random. The indistinguishable distributions are labelled $\mathcal{EEDH}^*$ and $\mathcal{EER}^*$. Which of these cases is appropriate will depend upon the protocol in question and its proof of security. Table 3 shows the requirements in each case, where the distributions are to be indistinguishable with a security level of k bits.

As an example putting it all together, suppose that a security level of $k = 80$ bits is required, we desire that the $\mathcal{EEDH}$ and $\mathcal{EER}$ distributions are indistinguishable, $q_1 = 1$ and $q_2 = 1$. Furthermore, suppose that m is prime. Then we need:

Table 3. Requirements for the two cases to be indistinguishable from random

For $\mathcal{EEDH}$ and $\mathcal{EER}$ indistinguishable:	For $\mathcal{EEDH}^*$ and $\mathcal{EER}^*$ indistinguishable:
– $\frac{S_5}{\epsilon_5} \geq 2^{k+1}$	– $\frac{S_5 - S_8}{\epsilon_5} \geq 2^{k+1} q_1$ where $S_8 \approx (q_1 - 1) q_2$
– $e \geq k + 2 + \log_2(q_1)$	– $e \geq k + 2 + 2\log_2(q_1)$
– t-DDH assumptions:	– t-DDH assumptions:
$\left(2^{k+4} q_1 + q_1 + q_2, \frac{1}{2}\right)$ and	$\left(2^{k+4} q_1^2 + q_1 q_2, \frac{1}{2}\right)$ and
$\left(q_1 + q_2 + 1, \frac{1}{2^{k+3} q_1}\right)$	$\left(q_1 q_2 + 1, \frac{1}{2^{k+3} q_1^2}\right)$
– s-DLSE assumptions:	– s-DLSE assumptions:
$\left(2^{i-1} s \ln(2s)(Y + 2Z), \frac{1}{2}\right)$ and	$\left(2^{i-1} s \ln(2s)(Y + 2Z), \frac{1}{2}\right)$ and
$\left(Y^i s \ln(s)(Z + 1), \frac{1}{Y}\right)$ where	$\left(Y^i s \ln(s)(Z + 1), \frac{1}{Y}\right)$ where
$Y \overset{\text{def}}{=} (\log_2(m) - s)\, 2^{k+5} q_1$,	$Y \overset{\text{def}}{=} (\log_2(m) - s)\, 2^{k+5} q_1^2$,
$Z \overset{\text{def}}{=} S_3 + q_1 + q_2$.	$Z \overset{\text{def}}{=} S_3 + q_1 q_2$.

In both cases $i = 3$ unless $\log_2(m) > 2s - \log_2(\epsilon_1)$, in which case $i = 2$ and the smallest sensible value for ϵ_1 is $\frac{1}{Y}$. S_3 is the cost of a multi-exponentiation in G.

- a randomness expander with an 81 bit security level, e.g. CBC-MAC with a 128 bit key for its block cipher or MD5 with a 128 bit key;
- a $(t, 2^{-82})$ strong randomness extractor for some t that outputs enough bits to key the randomness expander, e.g. a universal hash function—in that case $t = 292$ (see Sect. 4.2);
- $\left(2^{84}, \frac{1}{2}\right)$ and $\left(3, \frac{1}{2^{83}}\right)$ t-DDH assumptions on G, e.g. G could be of order 292 bits on an elliptic curve (292 is the maximum of $t = 292$ and $2 \cdot 85$);
- exponents of the full 292 bits since the short exponent assumption needs the short exponent to be longer than 292 bits (probably around 600 bits).

Further details of the calculations are provided in the full paper. This example contradicts the statement by Gennaro et al. [1, Sect. 6] that exponents of length $2k$ may be used to achieve a security level of k bits, since in our example, we need the exponent to be of length between $5k$ and $7.4k$. It seems that Gennaro et al. have not substituted actual values into their theorem stating that short exponents may be used, and have thus come to an incorrect conclusion about how long the short exponents really need to be.

4.2 Available Extractors

We now compare the available randomness extractors, focusing on output lengths of 128, 160, 256 and 512 bits, as these are the possible key lengths for the randomness expanders in Sect. 3.4. The reader may make his own comparisons for other output lengths with the information provided.

We first discuss the use of the Leftover Hash Lemma (LHL) to show that a universal (or almost universal) hash function may be used as a randomness extractor. Following this, we discuss the use of a PRFF as a randomness extractor, as analysed by Chevassut et al. [2], and then summarize the results of Fouque et al. [4] on deterministic extraction of lower order bits from subgroups of $\mathbb{Z}_p^*$. Then another work of Fouque et al. [5] is summarized with several results on using HMAC to extract randomness from the HMAC key, and a result on using the cascade construction as a randomness extractor. The full paper provides an overview and detailed comments on the problems with the first work [3] to consider the suitability of CBC-MAC, the cascade construction, and HMAC for use as randomness extractors in the standard model.

We aim for the output of the extractor to be (S', ϵ') indistinguishable from uniform with $\frac{S'}{\epsilon'} \geq 2^{81}$ as a minimum requirement (this will achieve a security level no greater than $k = 80$ bits when the randomness extractor and expander are used together). Table 3 will provide the basis for our numerical analysis of the advantages of each extractor. We will use the notation of Sect. 4.1 and assume (as was done there) that $S_4 \approx q_1$, $S_6 \approx q_2$ and $S_8 \approx (q_1 - 1)q_2$. Furthermore, we let c be the key length of the expander, and hence the output length of the extractor; t be the min-entropy, b be the block size and L be the number of blocks of the pre-secret (ps, e.g. the DH value) which is input to the extractor. We will examine the parameters required of each extractor to achieve various security levels in the following cases (notation is as in Sect. 4.1). In our examples, we use the cascade construction as the expander, since it is the best (see Sect. 3.4). The parameters required to achieve other security levels or in other cases can be derived by the reader.

1. Each extractor key is used only once ($q_1 = 1$; this would be the case if the key is chosen afresh in each protocol run); the expander is used only once or twice with each key ($q_2 \leq 2$); it is desired that $\mathcal{EEDH}$ and $\mathcal{EER}$ are indistinguishable (the KA protocol's security will be lower than k bits, since the total number of sessions will appear as a factor in its security reduction).
2. The extractor key is a global parameter used up to 2^{30} times ($q_1 \leq 2^{30}$); other requirements are as for the previous case; e.g. many other applications use the extractor at a k-bit security level; the KA protocol proof focuses on one session; that session's two keys (output by the expander) have k bits of security (again, the protocol's overall security will be lower than k bits).
3. Each extractor key is used once ($q_1 = 1$); the expander is used many times with each key ($q_2 > 2$); other requirements are the same as for the first case.
4. The extractor key is the same in all KA protocol sessions (but not used in other applications), and there are up to 2^{30} sessions ($q_1 \leq 2^{30}$); the expander is used many times with each key ($q_2 > 2$); $\mathcal{EEDH}^*$ and $\mathcal{EER}^*$ must be indistinguishable (so the number of sessions will not be an extra factor in the protocol proof). We assume $S_8^2 \approx q_1^2 q_2^2 \leq 2^{k+1} q_1$ so that a cascade construction security level of $k + 1 + \log_2(q_1)$ bits (less conservative option) gives $\frac{S_5 - S_8}{\epsilon_5} \geq 2^{k+1} q_1$.[2]

[2] We want $(S_5 - S_8)/\epsilon_5 \geq 2^{k+1} q_1$. When using the cascade construction (less conservative option) we have $1/\epsilon_5 \geq 2^c / \left(2^{20} S_5^2\right)$ (see the comments in the full paper), so we need $((S_5 - S_8)\, 2^c) / \left(2^{20} S_5^2\right) \geq 2^{k+1} q_1$ where c is the key length of the randomness extractor. When $S_5 = S_8 + 1$, we have $((S_5 - S_8)\, 2^c) / \left(2^{20} S_5^2\right) \geq 2^{k+1} q_1$ implies $2^c \geq 2^{k+21} q_1 S_8^2$. However, for a security level of s bits for the randomness expander, we require $2^c \geq 2^{2s+20}$, and if $s = k + 1 + \log_2(q_1)$, this will imply the first requirement when $s \geq 2 \log_2(S_8)$. For values of S_5 much larger than S_8, $S_5 - S_8 \approx S_5$ and so a security level of $k + 1 + \log_2(q_1)$ bits will be sufficient.

Almost Universal Hash Functions. The Leftover Hash Lemma (LHL) is well-known and allows the use of a universal (or almost universal) hash function as an extractor which is probabilistic and optimal in general [2]. There are several variations of the LHL in the literature; the one provided is mainly from Chevassut et al. [2], and similar to Dodis et al. [3, p.501].

Definition 13 (δ-AU (almost universal)). *Let c and b be integers, and $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ be a family of hash functions with domain $\{0,1\}^b$, range $\{0,1\}^c$ and key space $\mathcal{K}$. We say that the family $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is δ-almost universal (δ-AU)[3] if for every pair of different inputs x, y from $\{0,1\}^b$ it holds that $\Pr(h_\kappa(x) = h_\kappa(y)) \leq \delta$, where the probability is taken over $\kappa \in_{\mathrm{R}} \mathcal{K}$. For a given probability distribution $\mathcal{X}$ on $\{0,1\}^b$, we say that $\{h_\kappa\}_{\kappa\in\mathcal{K}}$ is δ-AU w.r.t. $\mathcal{X}$ if $\Pr(h_\kappa(x) = h_\kappa(y)) \leq \delta$ where the probability is taken over $\kappa \in_{\mathrm{R}} \mathcal{K}$ and $x, y \in_{\mathrm{R}} \mathcal{X}$ conditioned on $x \neq y$.*

An example of a universal hash function is the function that multiplies a Toeplitz matrix (one with constant diagonals) by the input to create the output [17]. The full paper gives more details and examples of universal hash functions.

Lemma 3 (LHL with δ-AU [2]). *Let $\mathcal{X}$ be a probabilistic distribution over $\{0,1\}^b$ with min-entropy at least t. Let e be an integer and $c \leq \alpha - 2e$ where $\alpha = \min(t, \log_2(1/\xi))$. Let $\mathcal{H} = \{h_\kappa\}_{\kappa\in\mathcal{K}}$, with h_κ having domain $\{0,1\}^b$ and range $\{0,1\}^c$ for any $\kappa \in \mathcal{K}$, be a δ-AU hash function family with $\delta = \frac{1}{2^c} + \xi$. Let H be a random variable uniformly distributed on $\mathcal{H}$, X denote a random variable taking values in $\{0,1\}^b$, and H and X be independent. Then, $(H, H(X))$ is 2^{-e}-uniform on $\mathcal{H} \times \{0,1\}^c$.*

This lemma states that a δ-almost universal hash function is a $(t, 2^{-e})$ strong randomness extractor. It was used to generate Table 4, where we must have $\xi \leq 2^{-t}$. It shows that even the most basic requirements mean a computational entropy of 292 bits in the input to the randomness extractor. More realistic requirements may mean a much higher level of computational entropy is required. Because of their significant key size requirements, and because other functions such as cryptographic hash functions are more readily available, universal hash functions are often not used for key derivation.

PRFFs as Randomness Extractors. Chevassut et al. [2] have shown that a PRFF may be used for randomness extraction with a publicly known key.

Theorem 1 ([2]). *If a family of functions, $\mathcal{F}$, is a $(S, 2, \xi)$-PRFF with domain $\{0,1\}^b$ and range $\{0,1\}^c$, S is the size of a circuit that makes 2 oracle queries on an instance of $\mathcal{F}$, then it is a $(\frac{1}{2^c} + \xi)$-AU hash function family.*

By using Lemma 3, we can conclude that a PRFF can be a strong randomness extractor, although the output of the PRF will generally need to be truncated to

[3] Being δ-AU in Dodis et al. [3] is the same as being ξ-AUH in Chevassut et al. [2] for $\delta = \frac{1}{2^c} + \xi$ where c is the number of bits of output of the function. We use the notation of Dodis et al. in this paper. When $\delta = \frac{1}{2^c}$, the function is universal.

Table 4. Universal hash function parameter examples

Case	t	k	e	c	Case	t	k	e	c
1	$c+2e$	k	$k+2$	$\geq (k+2)+2$	3	$c+2e$	k	$k+2$	$\geq 2(k+2)+20$
1	292	80	82	128	3	420	80	82	256
1	380	124	126	128	3	492	116	118	256
1	476	156	158	160	3	900	192	194	512
1	764	252	254	256	3	1004	244	246	512
1	1532	508	510	512					
2	$c+2e$	k	$k+32$	$\geq (k+32)+2$	4	$c+2e$	k	$k+62$	$\geq 2(k+32)+20$
2	352	80	112	128	4	540	80	142	256
2	476	126	158	160	4	552	86	148	256
2	704	192	224	256	4	956	160	222	512
2	764	222	254	256	4	1020	192	254	512
2	1532	478	510	512	4	1064	214	276	512

a length compatible with Lemma 3. Reuse of the extractor can then be covered by one of the theorems from the full paper. For example, to achieve a security level of $k = 80$ bits in Case 1, as shown in Table 4, we will need $\xi < 2^{-292}$. This rules out the use of CBC-MAC, since the block size is only likely to be 128 bits, and so the security level will only be about 125 bits. The use of HMAC or the cascade construction seems appropriate, provided we do not need ξ smaller than 2^{-508} or 2^{-510} respectively. In our example, we could use SHA-384 or better, and would need to truncate the output to 128 bits.

Deterministic Extraction of Lower Order Bits. The analysis of Fouque et al. [4] allows one to use the lower or higher-order bits from subgroups of $\mathbb{Z}_p^*$.

Theorem 2. *Let p be a b-bit prime, that is $2^{b-1} < p < 2^b$, G a subgroup of $\mathbb{Z}_p^*$ of order q with $q \gg \sqrt{p}$, l the integer such that $2^{l-1} \leq q \leq 2^l$ and X a random variable uniformly distributed in G. Let $\mathrm{lsb}_c(X)$ denote the c least significant bits of X. Let e be a positive integer and let $l > t = b/2 + c + e + \log_2(b) + 1$. Then the function $\mathrm{lsb}_c(\cdot)$ is a $(t, 2^{-e})$-deterministic extractor for the G-group distribution. If $p^{1/2} \leq q \leq p^{2/3}$ then the requirement on l may be refined to $l > t = b/4 + 3l/8 + c + e + \log_2(b) + 3$, and if $256 \leq q \leq p^{1/2}$, it may be refined to $l > t = b/8 + 5l/8 + c + e + \log_2(b) + 3$. Let $\mathrm{msb}_c(X)$ denote the c most significant bits of X and let $\delta = (2^n - p)/2^n$. If $3\delta < 2^{-e-1}$ and $l > t = n/2 + k + e + \log_2(n) + 1$, then $\mathrm{msb}_c(\cdot)$ is a $(t, 2^{-e})$-deterministic extractor.*

Table 5 shows some parameter examples using Theorem 2 with the four cases under consideration. Comparing it with Table 4, we can see that more computational entropy is generally required than when using a universal hash function. Fouque et al. recommended the use of the DLSE assumption to shorten the exponents required and thus improve efficiency. However, Sect. 4.1 indicates that much more than $2e$ bits will be required, contrary the indication of Fouque et al. (summarizing Gennaro et al.'s work [1]). However, one advantage of this method is that it is deterministic, and so does not require a key for the extractor.

Table 5. Parameter examples for least significant bits extraction

Case	b	t	k	e	c	Case	b	t	k	e	c
1			k	$k+2$	$\geq k+4$	3			k	$k+2$	$\geq 2(k+2)+20$
1	1024	733	80	82	128	3	1024	861	80	82	256
1	2048	1178	80	82	128	3	1024	897	116	118	256
1	1024	777	124	126	128	3	2048	1742	192	194	512
1	1024	841	156	158	160	3	2048	1794	244	246	512
1	2048	1546	252	254	256						
1	2048	2058	508	510	512						
2			k	$k+32$	$\geq k+34$	4			k	$k+62$	$\geq 2(k+32)+20$
2	1024	763	80	112	128	4	1024	921	80	142	256
2	1024	841	126	158	160	4	1024	927	86	148	256
2	1024	1003	192	224	256	4	2048	1770	160	222	512
2	2048	1546	222	254	256	4	2048	1802	192	254	512
2	2112	2091	478	510	512	4	2048	1824	214	276	512

HMAC. Fouque et al. [5] have analysed the security of HMAC as a randomness extractor when the data from which the randomness is to be extracted (pre-secret, *ps*) is used as the key of HMAC. Because the pre-secret is used as the HMAC key, some other data (denoted *label*, of at most l blocks), which is possibly adversarially generated, is used as the input to HMAC. There are two separate results, depending on whether the pre-secret is longer than one block or not.

Theorem 3 ([5]). *Using the notation of this section and Definition 9, let* $L = 1$, *let ipad and opad be chosen uniformly at random and let* IV *be a fixed string. Let* h' *be the hash function defined by* $h'_{IV}(pad, \cdot) = h(IV, \cdot \oplus pad)$ *where the key is pad. Let* S_h *be the circuit size for one computation of* h*. Let* h' *be a* $(S'+2S_h, q = 2, \epsilon_1)$ *PRFF, and* h *be both a* $(S', q = 1, \epsilon_2)$ *and* $(O(l \cdot S_h), q = 2, \epsilon_3)$ *PRFF. Then* $\mathrm{Hmac}_{IV}^{\mathrm{Hash}}(ipad, opad; ps, label)$ *is a* $(t, \infty, 0, S', \epsilon')$ *computational randomness extractor with* $\epsilon' \leq \frac{\sqrt{2^{2c}(2^{-t}+2\epsilon_1)}}{2} + \frac{1}{2^{c'}} + \epsilon_2 + 2l\epsilon_3$.

This is only useful if $b \gg 2c$, since $L = 1$ implies $t \leq b$ and when $t = 2c$ the term under the square root is at least one. In the case of SHA-1, we have $b = 512$ and $c = c' = 160$. To achieve a security level of e bits for the output of HMAC, we want $S'/\epsilon' \geq 2^e$. If we assume $\epsilon_1 \leq (S' + 2S_h)/(S_h 2^b)$, $\epsilon_2 \leq S'/(S_h 2^c)$, $\epsilon_3 \leq lS_h/(S_h 2^c)$, and $l \ll 2^c$, and consider the case where $S' = S_h = 1$, we require $e \leq \min\left(\frac{t-2c+1}{2}, \frac{b-2c+1.6}{2}, c', c - 2\log_2(l) - 1\right)$. These conditions will also ensure that the conditions placed on e when $S' = 2^{e-1}$, $S_h = 1$ and we want $\epsilon' \leq \frac{1}{2}$, are met. Hence, when $t = 512$, we achieve the maximum security level of $e = 96$ bits; for $e = 82$ bits, we need $t = 483$ bits min-entropy.

To overcome the problem of the above theorem only being useful when $b \gg 2c$, the assumptions on the compression function can be modified. That is, it is assumed that h is a PRFF resistant to related key attacks (RKA) when it is keyed with a bit string of min-entropy at least t (denoted t-RKA; $t = c$ for classical RKA). This assumption cannot be reduced to the h PRFF-security against RKA, since it is possible to have a good PRFF for a uniformly distributed key that is not a good PRFF for a high-entropy key. We omit the details of a RKA adversary used in the following theorems, but note that if the exhaustive search adversary with circuit size S' is the best known t-RKA adversary, its advantage is smaller than $(S'/S_h)/2^t$. Fouque et al. state their revised theorem in terms of HPRF, which is constructed from several concatenations and iterations of HMAC (they do not describe HPRF in detail but refer the reader to TLS v1.2 [18]).

Theorem 4 ([5]). *Let $L = 1$, let ipad and opad be two fixed strings and let IV be chosen uniformly at random. Let h be a function family resistant to a t-RKA adversary with circuit size S' that makes at most 2 queries with advantage ϵ_0. Let S_h be the circuit size for one computation of h. Let HPRF be a concatenation of v HMAC, and* Hash *be truncated. Let h be both a $(S', q = 2v, \epsilon_1)$ and $(O(l \cdot S_h), q = 2, \epsilon_2)$ PRFF. Then $\mathrm{Hprf}^{\mathrm{Hash}}_{ipad,opad}(IV; ps, label)$ is a $(t, \infty, 0, S', \epsilon')$ computational randomness extractor with $\epsilon' \leq \epsilon_0 + \epsilon_1 + 4v^2 l \epsilon_2 + \frac{2v^2}{2^{c'}} + \frac{v^2}{2^c}$.*

Assuming $l = v = 1$, $\epsilon_0 \leq \frac{S'/S_h}{2^t}$, $\epsilon_1 \leq \frac{(S'/S_h)}{2^c}$, and $\epsilon_2 \leq \frac{(lS_h/S_h)}{2^c}$, we have $S'/\epsilon' \geq 2^e$ when $e \leq t-3$, $e \leq c-5$ and $e \leq c'-4$. Hence, we can extract almost all of the pre-secret's entropy when it has less entropy than the number of bits output by HPRF, and the pre-secret is only one block long.

When the pre-secret is longer than one block, it is first hashed and padded with '0' bits to obtain a b-bit string. The following theorem covers this case for HMAC. We omit the similar theorem for HPRF (when it is constructed from several concatenations and iterations of HMAC) due to lack of space.

Theorem 5 ([5]). *Let $L \geq 2$, ipad and opad be fixed strings, and IV be a variable chosen uniformly at random. Define $\hat{h} : \{0,1\}^{c'} \times \{0,1\}^c \rightarrow \{0,1\}^c$ as $\hat{h}(x, y) = h(y, x \parallel 0^{b-c'})$. Let S_h be the circuit size for one computation of h. Let* Hash *be truncated. Let ϵ_2 be the RKA advantage of an adversary against $\hat{h}$ making at most 2 related key queries with circuit size S'. Let h be a $(S', q = 2, \epsilon_1)$, $(S', q = 1, \epsilon_3)$ and $(O(l \cdot S_h), q = 2, \epsilon_4)$ PRFF. Then $\mathrm{Hmac}^{\mathrm{Hash}}_{IV}(ipad, opad; ps, label)$ is a $(t, \infty, 0, S', \epsilon')$ computational randomness extractor with $\epsilon' \leq \frac{1}{2^{c'}} + \epsilon_2 + \epsilon_3 + 2l\epsilon_4 + \sqrt{2^{c'}\,(3 \cdot 2^{-t} + 2L\epsilon_1)}$.*

Assuming $\epsilon_1 \leq \frac{S'/S_h}{2^c}$, $\epsilon_2 \leq \frac{(S'/S_h)}{2^{c'}}$, $\epsilon_3 \leq \frac{(S'/S_h)}{2^c}$, and $\epsilon_4 \leq \frac{O(lS_h)/S_h}{2^c}$, we have $S'/\epsilon' \geq 2^e$ when $e \leq \frac{t-3.6-c'}{2}$, $e \leq \frac{c-c'-\log_2(L)-3}{2}$, $e \leq c'-2$ and $e \leq c-2\log_2(l)-3$. Hence, when $L = 2$, $l = 1$, and SHA-384 is used, only $e = 62$ bits of security can be achieved for the output of HMAC, and this requires $t \geq 512$. To achieve a value of e close to a value of c', we need $c' = e + 2 = \frac{c}{3}$. To achieve this we could further truncate the output of SHA-384 to only $c' = 170$ bits, and use this new hash function in the HMAC implementation. Then, provided $t \geq 510$, we would have $e = 168$. For $e > 168$, a new compression function h with output larger than 512 bits is needed. Alternatively, to achieve our minimum requirement for Case 1 described above, of $e = 82$ and $c' = 128$, we could use SHA-384 but further truncate the output to only $c' = 128$ bits. In that case we would only need $t \geq 296$ bits. This is similar to using a universal hash function, which is not surprising, since the analysis of Fouque et al. made use of the LHL.

Cascade Construction. Fouque et al. [5] also analysed the use of the cascade construction as a randomness extractor when the output is truncated to contain only c' bits, instead of c bits. Assume the compression function h of hash function H (with key IV) is an $(S, q = 2, \epsilon)$ PRFF. Then H is a $(t, \infty, 0, S', \epsilon')$ computational randomness extractor for prefix free distributions of at most L blocks with $S = O(S')$ and $\epsilon' \leq \sqrt{2^{c'} \cdot (3 \cdot 2^{-t} + 2L\epsilon)}$. As before, assume $\epsilon \leq S/2^c$. Hence, $\epsilon' \leq \sqrt{2^{c'} \cdot (3 \cdot 2^{-t} + 2^{1-c}L \cdot O(S'))}$. To achieve a security level of e bits for the output of H, we want $S'/\epsilon' \geq 2^e$. When $O(S') = 1$, this equates to requiring $\min\left(\frac{t-c'-3.6}{2}, \frac{c-c'-3-\log_2(L)}{2}\right) \geq e$. When the requirements for $O(S') = 1$ are met, those for when $O(S') = 2^{e-1}$ will be met also. These restrictions on e are almost the same as for HMAC when the pre-secret is more than one block long, and so similar comments to those made for HMAC apply here.

5 Conclusion

This paper examined the use of randomness extraction and expansion in key agreement protocols to generate uniformly distributed keys. Although other works exist that provide the basic theorems necessary, they lack details or examples of what cryptographic primitives are appropriate and/or how large the parameters of those primitives must be. We have therefore summarized existing work in the area and examined the security levels achieved with the use of various extractors and expanders for particular sizes of parameters.

As noted in some existing works ([1, p.4], [2, p.2]), the large amount of min-entropy needed in the pre-secret is often overlooked in efficiency comparisons of KA protocols. In fact, using the tables presented in this paper, one may conclude that this shared secret will need a min-entropy of at least 292 bits to achieve an overall security level of 80 bits. More realistic assumptions on the number of times the randomness extractor and expander are used may require a much higher min-entropy for this security level. The tables may be used to find the min-entropy required for various security levels and assumptions on how the extractor and expander will be used. We also found that when numbers are substituted into the short exponent theorems of Gennaro et al., the exponents may need to be much longer than they suggested.

References

1. Gennaro, R., Krawczyk, H., Rabin, T.: Secure hashed Diffie-Hellman over non-DDH groups. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 361–381. Springer, Heidelberg (2004), http://eprint.iacr.org/2004/099
2. Chevassut, O., Fouque, P.A., Gaudry, P., Pointcheval, D.: The Twist-AUgmented technique for key exchange. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 410–426. Springer, Heidelberg (2006), http://eprint.iacr.org/2005/061

3. Dodis, Y., Gennaro, R., Håstad, J., Krawczyk, H., Rabin, T.: Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 494–510. Springer, Heidelberg (2004)
4. Fouque, P.A., Pointcheval, D., Stern, J., Zimmer, S.: Hardness of distinguishing the MSB or LSB of secret keys in Diffie-Hellman schemes. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 240–251. Springer, Heidelberg (2006)
5. Fouque, P.A., Pointcheval, D., Zimmer, S.: HMAC is a randomness extractor and applications to TLS. In: ASIACCS 2008: Proceedings of the, ACM symposium on Information, computer and communications security, pp. 21–32. ACM, New York (2008)
6. Bellare, M., Kilian, J., Rogaway, P.: The security of the cipher block chaining message authentication code. Journal of Computer and System Sciences 61(3), 362–399 (2000), http://www-cse.ucsd.edu/~mihir/papers/cbc.html
7. Goldreich, O.: The Foundations of Cryptography, vol. 1. Cambridge University Press, Cambridge (2001), http://wisdom.weizmann.ac.il/~oded/frag.html
8. Shoup, V.: A Computational Introduction to Number Theory and Algebra. Cambridge University Press, Cambridge (2005), http://shoup.net/ntb/
9. Bellare, M., Canetti, R., Krawczyk, H.: Pseudorandom functions revisited: The cascade construction and its concrete security. In: Proceedings of the 37th Annual Symposium on the Foundations of Computer Science, pp. 514–523. IEEE, Los Alamitos (1996)
10. NIST (National Institute for Standards and Technology): Advanced encryption standard (AES). FIPS PUB 197 (2001)
11. Bellare, M.: New proofs for NMAC and HMAC: Security without collision-resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
12. Rivest, R.: The MD5 message-digest algorithm. Internet RFC 1321, Internet Engineering Task Force (1992)
13. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A strengthened version of RIPEMD. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 71–82. Springer, Heidelberg (1996)
14. NIST (National Institute for Standards and Technology): Secure hash standard. FIPS PUB 180-2 (2000)
15. Preneel, B., van Oorschot, P.: On the security of iterated message authentication codes. IEEE Transactions on Information Theory 45(1), 188–199 (1999)
16. Dodis, Y.: Exposure-Resilient Cryptography. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2000), http://theory.lcs.mit.edu/~yevgen/academic.html
17. Mansour, Y., Nisan, N., Tiwari, P.: The computational complexity of universal hashing. In: Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing—STOC 1990, pp. 235–243. ACM Press, New York (1990)
18. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) protocol version 1.2. Internet RFC 5246, Internet Engineering Task Force (2007)

Novel Precomputation Schemes for Elliptic Curve Cryptosystems

Patrick Longa and Catherine Gebotys

Department of Electrical and Computer Engineering,
University of Waterloo, Canada
{plonga,cgebotys}@uwaterloo.ca

Abstract. We present an innovative technique to add elliptic curve points with the form $P \pm Q$, and discuss its application to the generation of precomputed tables for the scalar multiplication. Our analysis shows that the proposed schemes offer, to the best of our knowledge, the lowest costs for precomputing points on both single and multiple scalar multiplication and for various elliptic curve forms, including the highly efficient Jacobi quartics and Edwards curves.

Keywords: Elliptic curve cryptosystem, scalar multiplication, multiple scalar multiplication, precomputation scheme, conjugate addition.

1 Introduction

In mid 80's, Miller and Koblitz independently proposed the use of elliptic curves for cryptographic purposes [8,16]. Since then, Elliptic Curve Cryptography (ECC) has gained increasing research and commercial interest. Scalar multiplication, denoted by kP, where k is a scalar and P is a point on the elliptic curve, is the central operation of most elliptic curve cryptosystems. A plethora of methods exist in the literature to execute this operation efficiently, mainly exploiting some efficient representation of the scalar. For instance, the Non-Adjacent Form (NAF) is a standard representation with the fewest nonzero terms using digits from the set $\{-1, 0, 1\}$.

In some settings, however, it is required to compute a multiple scalar multiplication with the form $kP + lQ$, where k and l are scalars and P and Q are points on the curve. In this scenario, well-known methods are Interleaving [17] and the Joint Sparse Form (JSF) [20].

A practical strategy that reduces further the number of required additions at the expense of some extra memory is the use of precomputations. In this case, a table of points is built and stored in advance (precomputation stage) for later use during the execution of the scalar multiplication itself (evaluation stage). Although these window-based methods effectively reduce the number of nonzero terms in most representations, a potential drawback is the cost of computing such a table, which grows with the window size.

Thus, it is an important research effort to minimize the cost of the precomputation stage to reduce the total cost of scalar multiplication. Further, although

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 71–88, 2009.

improved elliptic curve shapes with faster explicit formulae are currently the focus of intense research [1,6], there is still a lack of analysis of precomputation schemes that are efficient for these settings.

In that direction, this work proposes efficient precomputation schemes and analyzes their performance on *three* relevant elliptic curve settings: standard elliptic curves using Jacobian coordinates, Jacobi quartics using extended coordinates [6,7] and Edwards curves using inverted Edwards coordinates [2].

The proposed schemes are based on the following simple idea: if $P + Q$ has been computed for two distinct points P, Q, the subtraction of those points only requires a few additional field operations[1]. In the remainder, we will refer to this operation, namely $P - Q(= P + (-Q))$, as "conjugate" addition. It will turn out that this operation will allow computing precomputed tables very efficiently. We apply the strategy of the conjugate addition to calculate tables of the form d_iP and $c_iP \pm d_iQ$, which are commonly found in most single and multiple scalar multiplication algorithms.

Further, our precomputation schemes are compared and analyzed for *three* possible cases, which are determined by the system used to represent points: projective coordinates, affine coordinates with restriction to one inversion, and affine coordinates (without restriction in the number of inversions). Our extensive analysis allows determining which case is the most efficient for a particular scenario and for determined I/M (field inversion/multiplication) ratios.

Our work is organized as follows. In Section 2, we detail some background about ECC over prime fields. Then, in Section 3 we describe our strategy to derive low-cost formulas for the conjugate addition in the different settings under study. In Section 4, we introduce the new schemes for precomputing points for tables with the forms d_iP and $c_iP \pm d_iQ$, and discuss their costs. In Section 5, we analyze and compare the performance of the proposed schemes with the previously most efficient methods. A discussion of some other applications of the strategy of the conjugate addition follows in Section 6. Some conclusions summarizing the contributions of this work are presented at the end.

2 Preliminaries

An elliptic curve E over a prime field $\mathbb{F}_p$ is defined by the short Weierstrass equation E: $y^2 = x^3 + ax + b$, where $a, b \in \mathbb{F}_p$ and $\triangle = 4a^3 + 27b^2 \neq 0$, and which will be referred in the remainder as the standard elliptic curve form. The points on the curve E and the identity element $\mathcal{O}$, known as the point at infinity, form an abelian group whose group law essentially consists of two basic operations: doubling $(2P)$ and addition $(P + Q)$ of points.

The main operations in most elliptic curve-based cryptosystems have the forms kP and $kP + lQ$, known as (single) scalar multiplication and multiple scalar multiplication.

[1] Okeya et al. [18] showed that an inversion can be saved when computing $P \pm Q$ in affine coordinates. We expand the idea to projective coordinates for which further reductions are possible.

Affine coordinates (referred to as $\mathcal{A}$ in the remainder) uses (x, y) coordinates to represent points. However, since this system requires field inversions, it is generally expensive over prime fields. When using efficient forms for the prime p (as recommended by [4]), it has been observed that the cost of inversion can be as high as $1I > 30M$. For example, benchmarks by [10] and [3] show I/M ratios between 30-40 and 50-100, respectively.

In efficient implementations, point representations with the form $(X : Y : Z)$, known as projective coordinates, were introduced to replace inversions. For example, an efficient case of this projective representation is given by Jacobian coordinates (referred to as $\mathcal{J}$), where each projective point $(X_i : Y_i : Z_i)$ corresponds to the affine point $(X_i/Z_i^2, Y_i/Z_i^3)$. In this case, equation E acquires the form $Y^2 = X^3 + aXZ^4 + bZ^6$, and the negative of an element $P = (X_i, Y_i, Z_i)$ is given by $-P = (X_i, -Y_i, Z_i)$.

In recent years, other curve forms with faster group operations have appeared in the literature. In this work, we focus on two of them: Jacobi quartics and Edwards curves, whose explicit formulas have been found to be particularly fast. We briefly describe both curve shapes in the following. Note that we consider that constant curve parameters are fixed to small values so that the cost of performing any operation with them is negligible.

Jacobi quartic. It is defined by the curve $y^2 = x^4 + 2ax^2 + 1$, where $a \in \mathbb{F}_p$ and $a^2 \neq 1$. The projective curve is $Y^2 = X^4 + 2aX^2Z^2 + Z^4$, where a given projective point $(X_i : Y_i : Z_i)$ corresponds to the affine point $(X_i/Z_i, Y_i/Z_i^2)$. In this case, the negative of an element $P = (X_i, Y_i, Z_i)$ is represented by $-P = (-X_i, Y_i, Z_i)$. The most efficient formulae for these curves have been developed by Hisil et al. [6,7] using an extended coordinate system of the form $(X_i : Y_i : Z_i : X_i^2 : Z_i^2)$ that will be referred to as $\mathcal{JQ}$.

Edwards curve. It is defined by the curve $x^2 + y^2 = 1 + dx^2y^2$, where $d \notin \{0, 1\}$. In [1], Bernstein and Lange presented explicit formulas for point operations on this curve using standard projective coordinates. Later in [2], the same authors introduced a more efficient coordinate system, known as inverted Edwards coordinates (denoted by $\mathcal{IE}$), where each projective point $(X_i : Y_i : Z_i)$ corresponds to $(Z_i/X_i, Z_i/Y_i)$ in affine. In this case, the curve equation is given by $(X^2 + Y^2)\,Z^2 = X^2Y^2 + dZ^4$, where $XYZ \neq 0$, and the negative of a point $P = (X_i, Y_i, Z_i)$ is given by $-P = (-X_i, Y_i, Z_i)$.

In Table 1, we summarize the costs of the most efficient formulas in projective coordinates for the three curve forms under consideration. For complete details about formulas using $\mathcal{J}$ coordinates the reader is referred to [11,12]. Following the common practice in the literature, costs are expressed by the number of field multiplications (M) and squarings (S) that are required to perform certain operation, neglecting cheaper operations as field addition/subtraction (A) and multiplication/division by small constants. Table 1 includes efficient operations using mixed coordinates, which are useful if input point(s) are represented in affine ($\mathcal{A}$) coordinates but the result is required in some projective system $\mathcal{P}$. Also, note that we have included efficient formulas exploiting pre-stored values.

Table 1. Cost of elliptic curve point operations in projective coordinates using Jacobian ($\mathcal{J}$), inverted Edwards ($\mathcal{IE}$) and extended Jacobi quartic ($\mathcal{JQ}$) coordinates

Point Operation	Cost		
	Jacobian ($\mathcal{J}$; $a=-3$)	InvEdw ($\mathcal{IE}$)	JQuartic ($\mathcal{JQ}$)
Doubling (D), $2\mathcal{P} \to \mathcal{P}$	$3M+5S$	$3M+4S$	$2M+5S$
Mixed doubling (mD), $2\mathcal{A} \to \mathcal{P}$	$1M+5S$	$3M+3S$	$7S$
Tripling (T), $3\mathcal{P} \to \mathcal{P}$	$7M+7S$	$9M+4S$	$8M+4S$
Mixed tripling (mT), $3\mathcal{A} \to \mathcal{P}$	$5M+7S$	$7M+3S$	$5M+6S$
Addition $^{(1)}$ (A), $\mathcal{P}+\mathcal{P} \to \mathcal{P}$	$10M+4S$ / $9M+3S$	$-$	$7M+3S$
Addition (A), $\mathcal{P}+\mathcal{P} \to \mathcal{P}$	$11M+5S$	$9M+1S$	$7M+4S$
Mixed addition (mA), $\mathcal{P}+\mathcal{A} \to \mathcal{A}$	$7M+4S$	$8M+1S$	$6M+3S$
Mixed addition (mmA), $\mathcal{A}+\mathcal{A} \to \mathcal{A}$	$4M+2S$	$7M$	$4M+3S$
DA with stored values, $2\mathcal{P}+\mathcal{P} \to \mathcal{P}$	$13M+8S$	$-$	$-$
DA, $2\mathcal{P}+\mathcal{P} \to \mathcal{P}$	$14M+9S$	$-$	$-$
Mixed DA (mDA), $2\mathcal{P}+\mathcal{A} \to \mathcal{P}$	$11M+7S$	$-$	$-$

$\mathcal{P}$: projective coordinates ($\mathcal{J}$, $\mathcal{IE}$ or $\mathcal{JQ}$ coordinates)

(1) Addition with stored values.

If, for instance, values Z_1^2, Z_1^3, Z_2^2 and Z_2^3 are available when computing a general addition in $\mathcal{J}$ coordinates then we can saved up to $2M+2S$. Similarly, in the case of Jacobi quartics it is possible to reduce the original cost of $7M+4S$ of the addition formula to $7M+3S$ by noting that $(X_i+Z_i)^2$ can be precomputed (see [6] for more details).

Finally, Table 1 also includes the highly efficient doubling-addition operation (DA) developed by Longa and Miri in [13], which involves the recurrent operation $2P+Q$ and is more efficient than performing a traditional doubling followed by an addition using $\mathcal{J}$.

3 Our Strategy: Conjugate Addition

Our strategy to yield efficient precomputation schemes is based on the similarities between adding and subtracting two points. Basically, if the addition $P+Q$ takes place, then it is expected that, when subtracting the same points (i.e., $P-Q$), most of the intermediate field operations are identical simply because $P-Q=P+(-Q)$ and the negative of a point only involves the change of at most one of the coordinate values in the point representation, as described in the previous section.

Let us illustrate the latter with the point addition formula using $\mathcal{J}$. Let $P=(X_1,Y_1,Z_1)$ and $Q=(X_2,Y_2,Z_2)$ be two points on an elliptic curve E. If the addition $P+Q=(X_3,Y_3,Z_3)$ is performed using [12, formula (15)] as follows:

$$X_3=\alpha^2-(4\beta^3+8Z_2^2X_1\beta^2),\ Y_3=\alpha(Z_2^2X_1\beta^2-X_3)-Z_2^3Y_1\beta^3,\ Z_3=\theta\beta \quad (1)$$

where $\alpha=2(Z_1^3Y_2-Z_2^3Y_1)$, $\beta=Z_1^2X_2-Z_2^2X_1$ and $\theta=(Z_1+Z_2)^2-Z_1^2-Z_2^2$, then $P-Q$ can be computed as $P+(-Q)=(X_1,Y_1,Z_1)+(X_2,-Y_2,Z_2)=(X_4,Y_4,Z_4)$ reusing the partial values $(4\beta^3+8Z_2^2X_1\beta^2)$, $Z_2^2X_1\beta^2$, $-Z_2^3Y_1\beta^3$, Z_3, $Z_1^3Y_2$ and

$Z_2^3 Y_1$. The latter can be performed with the following formula for the conjugate addition:

$$X_4 = \gamma^2 - (4\beta^3 + 8Z_2^2 X_1 \beta^2),\ Y_4 = \gamma(Z_2^2 X_1 \beta^2 - X_4) - Z_2^3 Y_1 \beta^3,\ Z_4 = Z_3 \quad (2)$$

where $\gamma = -2(Z_1^3 Y_2 + Z_2^3 Y_1)$. Note that the cost of the conjugate addition (2) using $\mathcal{J}$ is only $1M + 1S$, which is significantly less than the cost of a general addition (1) (i.e., $11M + 5S$). If we also consider other usually neglected operations, then the cost drops from $11M + 5S + 9A + 2(\times 2) + 1(\times 4)$ to only $1M + 1S + 4A + 1(\times 2)$.

It may seem that performing this conjugate operation would involve several extra registers to store partial values temporarily. However, memory requirements can be minimized by performing $P + Q$ and $P - Q$ concurrently. For instance, a possible 35-step execution sequence for computing $P \pm Q$ using formulas (1) and (2) would be as the one shown in Table 2.

The execution of the addition/conjugate addition pair shown in Table 2 requires 8 registers only (including temporary registers and registers storing input coordinates). It is easy to verify that the memory requirement is the same as that of the addition formula alone. Thus, executing the conjugate addition does not increase the memory requirements in this case.

We have derived the conjugate addition formulas in projective coordinates (i.e., $\mathcal{J}$, $\mathcal{JQ}$ and $\mathcal{IE}$ coord.), and also in affine for the three curves of interest. The costs of these new formulas are summarized in Table 3. We have also included the

Table 2. Pseudocode of an "interlaced" execution of an addition/conjugate addition pair in $\mathcal{J}$ coordinates

```
INPUT: $T_1 \leftarrow X_1, T_2 \leftarrow Y_1, T_3 \leftarrow Z_1, T_4 \leftarrow X_2, T_5 \leftarrow Y_2, T_6 \leftarrow Z_2$
OUTPUT: $T_1 \leftarrow X_3, T_2 \leftarrow Y_3, T_3 \leftarrow Z_3, T_4 \leftarrow X_4, T_5 \leftarrow Y_4$
 1. $T_7 = T_3^2$             $\{Z_1^2\}$
 2. $T_4 = T_4 \times T_7$    $\{Z_1^2 X_2\}$
 3. $T_8 = T_3 \times T_7$    $\{Z_1^3\}$
 4. $T_5 = T_5 \times T_8$    $\{Z_1^3 Y_2\}$
 5. $T_8 = T_6^2$             $\{Z_2^2\}$
 6. $T_7 = T_7 + T_8$         $\{Z_1^2 + Z_2^2\}$
 7. $T_3 = T_3 + T_6$         $\{Z_1 + Z_2\}$
 8. $T_3 = T_3^2$             $\{(Z_1 + Z_2)^2\}$
 9. $T_3 = T_3 - T_7$         $\{\theta\}$
10. $T_6 = T_6 \times T_8$    $\{Z_2^3\}$
11. $T_2 = T_2 \times T_6$    $\{Z_2^3 Y_1\}$
12. $T_8 = T_1 \times T_8$    $\{Z_2^2 X_1\}$
13. $T_7 = T_4 - T_8$         $\{\beta\}$
14. $T_3 = T_3 \times T_7$    $\{Z_3 = Z_4\}$
15. $T_3 = T_7^2$             $\{\beta^2\}$
16. $T_6 = T_6 \times T_7$    $\{\beta^3\}$
17. $T_8 = T_6 \times T_8$    $\{Z_2^2 X_1 \beta^2\}$
18. $T_4 = 2T_8$              $\{2Z_2^2 X_1 \beta^2\}$
19. $T_4 = T_4 + T_7$         $\{\beta^3 + 2Z_2^2 X_1 \beta^2\}$
20. $T_4 = 4T_4$              $\{4\beta^3 + 8Z_2^2 X_1 \beta^2\}$
21. $T_6 = T_5 - T_2$         $\{Z_1^3 Y_2 - Z_2^3 Y_1\}$
22. $T_6 = 2T_6$              $\{\alpha\}$
23. $T_5 = -T_5 - T_2$        $\{-(Z_1^3 Y_2 + Z_2^3 Y_1)\}$
24. $T_5 = 2T_5$              $\{\gamma\}$
25. $T_1 = T_6^2$             $\{\alpha^2\}$
26. $T_1 = T_1 - T_4$         $\{X_3\}$
27. $T_7 = T_2 \times T_7$    $\{Z_2^3 Y_1 \beta^3\}$
28. $T_2 = T_8 - T_1$         $\{Z_2^2 X_1 \beta^2 - X_3\}$
29. $T_2 = T_2 \times T_6$    $\{\alpha(Z_2^2 X_1 \beta^2 - X_3)\}$
30. $T_2 = T_2 - T_7$         $\{Y_3\}$
31. $T_6 = T_5^2$             $\{\gamma^2\}$
32. $T_4 = T_6 - T_4$         $\{X_4\}$
33. $T_8 = T_8 - T_4$         $\{Z_2^2 X_1 \beta^2 - X_4\}$
34. $T_8 = T_5 \times T_8$    $\{\gamma(Z_2^2 X_1 \beta^2 - X_4)\}$
35. $T_5 = T_8 - T_7$         $\{Y_4\}$
```

Table 3. Costs of new conjugate additions for standard, Edwards and Jacobi quartic curves using projective ($\mathcal{J}$, $\mathcal{IE}$ and $\mathcal{JQ}$) and affine coordinates

Point Operation	Cost		
	Standard curve	Edwards curve	Jacobi quartic
Conjugate addition (A'), $\mathcal{P} - \mathcal{P} \rightarrow \mathcal{P}$	$1M + 1S$	$4M$	$2M + 1S$
Addition (A), $\mathcal{P} + \mathcal{P} \rightarrow \mathcal{P}$	$11M + 5S$	$9M + 1S$	$7M + 3S$
Conjug. mixed addition (mA'), $\mathcal{P} - \mathcal{A} \rightarrow \mathcal{P}$	$1M + 1S$	$4M$	$2M + 1S$
Mixed addition (mA), $\mathcal{P} + \mathcal{A} \rightarrow \mathcal{P}$	$7M + 4S$	$8M + 1S$	$6M + 3S$
Conjug. mixed addition (mmA'), $\mathcal{A} - \mathcal{A} \rightarrow \mathcal{P}$	$1M + 1S$	$3M$	$1M + 1S$
Mixed addition (mmA), $\mathcal{A} + \mathcal{A} \rightarrow \mathcal{P}$	$4M + 2S$	$8M$	$5M + 3S$
Conjugate addition (A'), $\mathcal{A} - \mathcal{A} \rightarrow \mathcal{A}$	$2M + 1S$	$4M$	$3M$
Mixed addition (A), $\mathcal{A} + \mathcal{A} \rightarrow \mathcal{A}$	$1I + 2M + 1S$	$1I + 9M + 1S$	$1I + 7M + 4S$

$\mathcal{P}$: projective coordinates ($\mathcal{J}$, $\mathcal{IE}$ or $\mathcal{JQ}$ coordinates).

costs of the traditional addition operations that accompany the execution of our formulas. Note that, in some cases, the traditional operations have been modified slightly so that the cost of the pair addition/conjugate addition is minimized. Refer to Appendices A-C for complete details.

As it can be seen in Table 3, the new conjugate formulas introduce significant cost reductions in comparison to traditional operations (see Table 1). In the following section, we take advantage of the latter to develop low-cost precomputation schemes.

4 New Precomputation Method for Scalar Multiplication

In this Section, we apply the concept of conjugate addition to derive highly efficient precomputation schemes first for tables of the form d_iP and then for tables of the form $c_iP \pm d_iP$. We consider *three* possible scenarios: precomputed points are left in projective coordinates (referred to as *case 1*), precomputed points are calculated in projective coordinates and then converted to affine using one inversion (referred to as *case 2*), and precomputed points are computed and left in affine (referred to as *case 3*).

4.1 Precomputation Scheme for Table of the Form d_iP

Well-known methods to compute scalar multiplication using a precomputed table with points d_iP, where $d_i \in \{3, 5, \ldots, m\}$, are Window-$w$ NAF (wNAF) and Fractional Window-w NAF (Frac-wNAF), in the case of single scalar multiplication, and the Interleaving method, in the case of multiple scalar multiplication.

We propose a recursive scheme that first tries to reach a "strategic" point and then applies efficiently the conjugate addition technique described in Section 3. In the following, we define as "strategic" to those points that can be efficiently computed and from which it is possible to calculate the maximum possible number of precomputed points at the lowest cost. The steps of our scheme are detailed in the following.

Step 1: Computation of precomputed points. This is the main body of our scheme, and is presented in Algorithm 4.1. In this step, points can be computed in projective coordinates using operations from Table 1 (case 1), or directly in $\mathcal{A}$ (case 3). If projective points are to be converted to $\mathcal{A}$ (case 2), then Step 2 should be executed right after.

Algorithm 4.1. Computation of precomputed points

```
Input: a point $P$ in affine ($\mathcal{A}$) coordinates, and an odd value $m \geq 7$
        to build a table of the form $d_iP$, where $d_i \in \{3, 5, 7, \ldots, m\}$
Output: the table $T = \{T_1 = 3P, \ldots, T_{(m-1)/2} = mP\}$ in projective or $\mathcal{A}$ coord.
1:  $r = 3$, $l = 1$, $i = 2$, $n = v = 0$
2:  $T_0 = P$, $T_1 = rP$
3:  $R = T_1$
4:  While $n < (m-3)/2$ do
5:    If $m < 2r$
6:      While $n < (m-3)/2$ do
7:        $T_s = R + T_l$
8:        $n = n + 1$, $l = l + 1$, $s = s + 1$
9:    Else
10:     $t = 2^v$
11:     $v = v + 1$
12:     $R = 2R$
13:     $r = 2r$, $j = t - 1$, first $= 1$
14:     While $j \geq 0$ do
15:       $T_i = R - T_j$, $n = n + 1$
16:       If first=1, then $l = j + 1$, $s = r - i$, first $= 0$
17:       $i = i + 1$
18:       If $m \geq r + 2j + 1$, then
19:         $T_{(r+2j)/2} = R + T_j$, $n = n + 1$
20:         If $T_j = T_0$, then $i = i + 1$
21:       $j = j - 1$
22: Return $T = \{T_1 = 3P, \ldots, T_{(m-1)/2} = mP\}$
```

Basically, Algorithm 4.1 first reaches certain "strategic" point and then computes all the points that are close to it by efficiently performing additions and conjugate additions. The "strategic" points proposed in our scheme have the form $P_{i+1} = 2P_i$, for $i \in \mathbb{Z} \geq 0$ and $P_0 = 3P$ (i.e., $6P, 12P, 24P$, and so on), which are computed using a combination of one tripling (performed at the beginning, Step 2) and a sequence of doublings (Step 12). Note that there is a minimum number of close points that makes the computation of a "strategic" point worthwhile. If that minimum is not fulfilled (evaluation in Step 5) then the algorithm calculates the remaining points from the previous "strategic" point (loop beginning in Step 6). The value of such a minimum depends on the particular costs of point operations. For $\mathcal{J}$, $\mathcal{JQ}$ and $\mathcal{IE}$, we have determined that the lowest cost is achieved if the next "strategic" point is computed always that the m value is greater or equal to such a "strategic" point (condition in Step 5).

Let us illustrate the proposed scheme with the following example.

Example 1. If $m = 13$, Alg. 4.1 computes the first points as $P \rightarrow 3P \rightarrow 6P$, where $6P$ is the first "strategic" point. From this, $5P$ and $7P$ (*close* points) are calculated by adding $6P+(-P)$ and $6P+P$. Note that the latter operation can be calculated with a conjugate addition, requiring a very low number of operations. Then, Alg. 4.1 calculates the following "strategic" point (since $m > 12$) by doubling $6P \rightarrow 12P$, and finally computes close points $9P$, $11P$ and $13P$ by performing $12P + (-3P)$, $12P + (-P)$ and $12P + P$, respectively. Note again that the latter operation is also a low-cost conjugate addition.

In Appendix D, we have sketched the derivation of points for tables with different values m. Note that the method described does not include cases $m = 3, 5$. Computing the table for $m = 3$ only requires *one* mixed tripling. For case $m = 5$, $\mathcal{JQ}$ and $\mathcal{J}$ coordinates, it is more efficient to compute points by performing $P \rightarrow 2P \rightarrow 4P$, and then obtaining $3P$ and $5P$ with an addition/conjugate addition pair (i.e., $4P + (-P)$ and $4P + P$). For case $\mathcal{IE}$, we suggest to compute the table following the sequence $P \rightarrow 2P \rightarrow 3P \rightarrow 5P$.

In the following, we describe the procedure to convert points to $\mathcal{A}$ for case 2.

Step 2: Conversion to affine (if required). If mixed addition (or mixed DA) is significantly more efficient than general addition (or general DA) in a given setting, then it would be convenient to express the precomputed table in $\mathcal{A}$.

It is known that conversion to $\mathcal{A}$ can be achieved by calculating $(X_i/Z_i^2, Y_i/Z_i^3)$, $(X_i/Z_i, Y_i/Z_i^2)$ and $(Z_i/X_i, Z_i/Y_i)$ for $\mathcal{J}$, $\mathcal{JQ}$ and $\mathcal{IE}$, respectively.

For each setting, calculation of denominators (denoted by u_i) can be efficiently carried out by using the well-known Montgomery' simultaneous inversion method so that the number of expensive inversions is limited to only *one*.

First, we compute the inverse $U = (u_1 u_2 \ldots u_t)^{-1}$, where u_i are all distinct denominators of the expressions above from all the non-trivial points in the table $\{3P, 5P, \ldots, mP\}$. For $\mathcal{J}$ and $\mathcal{JQ}$, the number of such denominators is reduced to only $t = (m-1)/2 - c$, where c is the number of points computed via conjugate addition, since points computed with addition/conjugate addition pairs share the same coordinate Z (see Appendices A-B). For $\mathcal{IE}$, $t = m - 1$ as each point has two distinct denominators, namely X_i and Y_i.

Then, individual denominators u_i are recovered from U, and the results multiplied to their corresponding numerator following the conversion expressions.

As it can be seen the use of conjugate additions reduces the cost of the Montgomery's method for the cases of $\mathcal{J}$ and $\mathcal{JQ}$ coordinates. Following our explanation above, it can be easily verified that one saves $3M + 1S$ per point computed with a conjugate addition.

Cost Analysis. The cost of the scheme proposed mainly depends on the value m in the precomputed table and the curve form selected. We list in Table 4 the costs in terms of number of operations for various values m. As operations in $\mathcal{A}$

Table 4. Cost of the proposed precomputation scheme: case 1 in projective coordinates using $\mathcal{J}$ and $\mathcal{JQ}$; case 2 using one inversion; and case 3 in $\mathcal{A}$

m	Point Operation Count	Case 1		Case 2	
		$\mathcal{J}$	$\mathcal{JQ}$	$\mathcal{J}$	$\mathcal{JQ}$
7	1mT+1D+1mA+1mA'	$17M + 17S$	$15M + 17S$	$1I + 28M + 18S$	$1I + 24M + 20S$
9	1mT+1D+1mA+1mA'+1A	$27M + 21S$	$22M + 20S$	$1I + 43M + 22S$	$1I + 36M + 25S$
11	1mT+1D+1mA+1mA'+2A	$37M + 25S$	$29M + 23S$	$1I + 59M + 27S$	$1I + 48M + 30S$
13	1mT+2D+2mA+2mA'+1A	$39M + 31S$	$32M + 30S$	$1I + 63M + 32S$	$1I + 53M + 35S$
15	1mT+2D+2mA+2mA'+1A+1A'	$40M + 32S$	$34M + 32S$	$1I + 67M + 33S$	$1I + 57M + 37S$

m	Point Operation Count	Case 3 Standard curve
7	1T+1D+1A+1A'	$3I + 13M + 7S$
9	1T+1D+1A+1A'+1A	$4I + 15M + 8S$
11	1T+1D+1A+1A'+2A	$5I + 17M + 9S$
13	1T+2D+2A+2A'+1A	$6I + 21M + 11S$
15	1T+2D+2A+2A'+1A+1A'	$6I + 23M + 12S$

coord. are relatively expensive in Jacobi quartic and Edwards curves (see Table 3), we only show the performance of case 3 in the setting of the standard curve.

Depending on the curve form selected, some additional considerations are necessary. In the case of the standard curve using $\mathcal{J}$, if the evaluation stage uses the efficient addition with *two* stored values, then values Z_i^2 and Z_i^3 should be computed during the precomputation stage. Naively, the latter would require $(1M + 1S)(m - 1)/2$. However, some additional cost reductions are possible. First, the initial tripling computes the required values for point $3P$ (i.e., Z_{3P}^2 and Z_{3P}^3) without requiring extra operations. Also, one squaring can be saved every time a doubling is performed to get any "strategic" point since values Z_i^2 are cached. Moreover, it is easy to see that addition and conjugate addition formulas share the same coordinate Z (see Appendix A). Hence, we only require $1M+1S$ to get Z_i^2 and Z_i^3 for two points computed with an addition/conjugate addition pair. Finally, when performing additions using a "strategic" point Q, its values Z_Q^2 and Z_Q^3 are calculated in the first mixed addition, say $Q + P = (X_Q, Y_Q, Z_Q) + (x_1, y_1)$. Thus, following general additions of the form $Q + R = (X_Q, Y_Q, Z_Q) + (X_R, Y_R, Z_R)$ can be executed using an addition with *four* stored values, taking into account that R is a point from the table and that values Z_R^2 and Z_R^3 are, hence, precalculated.

Similarly, in $\mathcal{JQ}$, if the evaluation stage uses the efficient addition with the stored value $(X_i + Z_i)^2$, then these values should be included in the precomputation cost. We now describe a few optimizations to minimize this cost. First, one squaring can be saved every time a doubling is performed to get any "strategic" point by noting that $(X_i + Z_i)^2$ can be cached from a previous mixed tripling or mixed addition. Also, when performing additions with a "strategic" point Q, the value $(X_Q + Z_Q)^2$ is calculated in the first mixed addition. Then, following general additions with the same point Q save one extra squaring.

The costs including the savings described above are detailed in Table 4, case 1. For the case where points are converted to $\mathcal{A}$ (case 2), we have to also consider the cost of performing the Montgomery' simultaneous inversion method (Step 2). The cost of the latter in $\mathcal{J}$ and $\mathcal{JQ}$ is given by $Cost_{\mathcal{J}\rightarrow\mathcal{A}} = 1I + (6L - 3)\,M + (L)\,S$ and $Cost_{\mathcal{JQ}\rightarrow\mathcal{A}} = 1I + (5L - 3)\,M + (2L)\,S$, respectively, where $L = (m-1)/2$ and m odd ≥ 5. However, as described in Section 4.1, Step 2, the proposed scheme allows for some extra savings since points obtained through an addition/conjugate addition pair share the same coordinate Z. The reduced costs including these savings are given by

$$Cost_{proposed\,\mathcal{J}\rightarrow\mathcal{A}} = 1I + (6L - 3c - 3)\,M + (L - c)\,S \tag{3}$$

$$Cost_{proposed\,\mathcal{JQ}\rightarrow\mathcal{A}} = 1I + (5L - 3c - 3)\,M + (2L - c)\,S \tag{4}$$

respectively, where c denotes the number of points obtained using a conjugate addition. In the case of $\mathcal{IE}$, the cost of the Montgomery's method is as follows

$$Cost_{\mathcal{IE}\rightarrow\mathcal{A}} = 1I + (6L + \lceil (L-2)/L \rceil - 1)\,M \tag{5}$$

The total costs including conversion to $\mathcal{A}$ are given in Table 4, case 2. Note that in this case addition operations with stored values do not apply.

4.2 Precomputation Scheme for Table of the Form $c_iP \pm d_iQ$

This scenario mainly applies to methods for computing multiple scalar multiplications such as those based on JSF [20]. In this case, the application of our strategy of conjugate additions is straightforward since precomputed points have the form $c_iP \pm d_iQ$, where $c_i, d_i \in \{0, 1, 3, 5, \ldots, m\}$, and each two points $cP \pm dP$ having $c, d \neq 0$ can be computed with an addition/conjugate addition pair.

In the following, we analyze the cost involved when precomputing points for the specific case of the efficient JSF-based algorithm by Kuang et al. [9]. Extension of the method to similar table forms easily follows.

Cost Analysis. If P and Q are unknown before the scalar multiplication is executed, the points $3P$,$3Q$,$P{\pm}Q$,$3P{\pm}Q$,$P{\pm}3Q$,$3P{\pm}3Q$ required by the method by [9] need to be computed on the fly. The latter costs 2mT+2mmA+4mA+2A for case 1 (when points are left in projective coord.). With the strategy of conjugate additions, that cost reduces to 2mT+1mmA+1mmA'+2mA+2mA'+1A+1A'. Note that the advantage increases for case 2 as our approach allows saving some operations during conversion to $\mathcal{A}$, as shown in Section 4.1.

In Table 5, we show the cost performance of the proposed scheme for the considered curve shapes. Note that, in the setting of $\mathcal{J}$ and $\mathcal{JQ}$, we use again the efficient addition formulas with stored values and, following the same procedure described in Section 4.1, we have minimized the impact of the computation of those partial values for case 1. For case 2 the conversion to $\mathcal{A}$ coordinates is similar to that of the scheme from Section 4.1 and, hence, it follows the costs given by (3), (4) and (5) for $\mathcal{J}$, $\mathcal{JQ}$ and $\mathcal{IE}$, respectively. Again, as operations in affine are relatively expensive in Edwards and Jacobi quartic curves, we only show the performance of case 3 in the setting of standard curves.

Table 5. Cost of the proposed precomputation scheme for the JSF$_3$ method [9]: case 1 in projective coord. using $\mathcal{J}$ and $\mathcal{JQ}$; case 2 using one inversion; and case 3 in $\mathcal{A}$.

Curve form	Point operations	Case 1	Case 2	Case 3
Jacobi quartic ($\mathcal{JQ}$)	2mT+1mmA+1mmA'+ 2mA+2mA'+1A+1A'	$41M + 35S$	$1I + 76M + 44S$	–
Edwards ($\mathcal{IE}$)		$47M + 24S$	$1I + 107M + 24S$	–
Standard ($\mathcal{J}$)		$42M + 32S$	$1I + 84M + 35S$	$6I + 30M + 16S$

5 Performance Comparison

In this section, we analyze and compare the proposed approach with the most efficient precomputation schemes available in the literature.

In the case of $\mathcal{J}$, Longa and Miri [13] recently proposed a highly efficient scheme, which has been shown to achieve the lowest cost among methods using only one inversion (case 2). The cost of this method (referred to as LM method in the remainder) is given by ($1M = 0.8S$)

$$Cost_{LM,\, case2} = \; 1I + (9L)\,M + (2L+6)\,S = 1I + (10.6L + 4.8)\,M \qquad (6)$$

We now derive the cost of the LM method for case 1 using the traditional chain $P \to 2P \to 3P \to 5P \to \ldots \to mP$ and the special addition due to [15], but avoiding the final conversion to $\mathcal{A}$. This involves one mixed doubling and L special additions that cost $5M + 2S$. Also, the use of additions with pre-stored values during the evaluation stage requires precalculating values Z_i^2 and Z_i^3 with a cost of $L(1M + 1S)$. Then the total cost is

$$Cost_{LM,\, case1} = \; (6L+1)\,M + (3L+5)\,S = \; (8.4L + 5)\,M \qquad (7)$$

Regarding $\mathcal{IE}$ and $\mathcal{JQ}$, we could not find any literature related to precomputation schemes in these settings. Hence, we analyze in the following the performance of the straightforward implementation using the traditional chain given above.

The cost of precomputation without using inversions (case 1) is given by

$$Cost_{\mathcal{IE},\, case1} = \; (9L+2)\,M + (1L+3)\,S = (9.8L + 4.4)\,M \qquad (8)$$

$$Cost_{\mathcal{JQ},\, case1} = \; (7L-1)\,M + (3L+7)\,S = (9.4L + 4.6)\,M \qquad (9)$$

for $\mathcal{IE}$ and $\mathcal{JQ}$ coordinates, respectively. These costs are derived by adding the costs of performing one mixed doubling, one mixed addition and $(L-1)$ general additions. For case 2, the costs are given by

$$Cost_{\mathcal{IE},\, case2} = \; 1I + (15.8L + \lceil (L-2)/L \rceil + 3.4)\,M \qquad (10)$$

$$Cost_{\mathcal{JQ},\, case2} = \; 1I + (12L-4)\,M + (5L+7)\,S = 1I + (16L + 1.6)\,M \qquad (11)$$

which are derived by adding the cost of performing the Montgomery's method of simultaneous inversion to equations (8) and (9).

Table 6. Costs of various schemes in projective (case 1) and affine (case 2); $1M = 0.8S$

Case	Method	Curve form	$w = 3$	$w = 4$	$w = 5$	$w = 6$
case 1	Proposed scheme	$\mathcal{JQ}$	$10.6M$	$28.6M$	$59.6M$	$116.6M$
	Method (9)	$\mathcal{JQ}$	–	$32.8M$	$70.4M$	$145.6M$
	Proposed scheme	$\mathcal{IE}$	$9.4M$	$28.4M$	$61.2M$	$121.6M$
	Method (8)	$\mathcal{IE}$	–	$33.8M$	$73.0M$	$151.4M$
	Proposed scheme	$\mathcal{J}$	$10.6M$	$30.6M$	$65.6M$	$130.6M$
	LM Method (7)	$\mathcal{J}$	–	$30.2M$	$63.8M$	$131.0M$
case 2	Proposed scheme	$\mathcal{JQ}$	–	$1I + 40.0M$	$1I + 86.6M$	$1I + 173.6M$
	Method (11)	$\mathcal{JQ}$	–	$1I + 49.6M$	$1I + 113.6M$	$1I + 241.6M$
	Proposed scheme	$\mathcal{IE}$	–	$1I + 46.4M$	$1I + 103.2M$	$1I + 211.6M$
	Method (10)	$\mathcal{IE}$	–	$1I + 46.8$	$1I + 102.0M$	$1I + 212.4M$
	Proposed scheme	$\mathcal{J}$	$1I + 10.2M$	$1I + 42.4M$	$1I + 93.4M$	$1I + 194.0M$
	LM Method (6), [13]	$\mathcal{J}$	–	$1I + 36.6M$	$1I + 79.0M$	$1I + 163.8M$

In Table 6, we compare the costs of the described schemes to that of the proposed scheme from Section 4.1 for different windows w. Costs for the latter method are derived from Table 4 and Appendix D. As it can be seen, the new approach outperforms every other method in cases 1 and 2 for both $\mathcal{IE}$ and $\mathcal{JQ}$. Note that the advantage increases with the window size. For instance, if $1I = 30M$, the cost reduction can be as high as 25% ($w = 6$, $\mathcal{JQ}$). Nevertheless, in case 2 with $\mathcal{IE}$ coordinates, both the proposed and traditional methods offer comparable performance.

In the case of standard curves, the LM scheme still achieves the highest performance. Nevertheless, for case 1, the modified LM scheme (7) and the new approach achieve similar performance.

In settings where inversions are not so expensive (low I/M ratios), it could be attractive the implementation of case 3. In this case, Table 7 shows the performance of the traditional approach and the proposed method on a standard curve form. Also, the I/M ratios for which the traditional, the proposed and the LM method achieve the lowest cost are shown at the bottom of the table. As it can be observed, the LM method offers the highest performance for a wide range of high I/M ratios on a standard curve, whereas the proposed method is convenient for low/intermediate values I/M.

Table 7. Costs of different schemes in affine (case 3) and I/M ranges for which each scheme achieves the lowest cost on a standard curve; $1M = 0.8S$

Method	$w = 4$	$w = 5$	$w = 6$
Proposed scheme, case 3	$3I + 19.4M$	$6I + 34.2M$	$11I + 60.2M$
Traditional	$4I + 12.0M$	$8I + 23.2M$	$16I + 45.6M$
I/M range (LM Method (6), [13])	$I > 8.6M$	$I > 9M$	$I > 10.4M$
I/M range (Proposed, case 3)	$7.4M < I < 8.6M$	$5.5M < I < 9M$	$2.9M < I < 10.4M$
I/M range (Traditional)	$I < 7.4M$	$I < 5.5M$	$I < 2.9M$

Let us now compare the performance of our scheme for cases 1 and 2, to determine the best scheme for each scenario. For this analysis, we should also consider the scalar multiplication cost since different point operations apply to different cases. Note that we only analyze the performance on Edwards and Jacobi quartic curves, as these are the settings where our method has been shown to attain the lowest costs (see Table 6).

Let us consider the standard wNAF method. In this case, the cost of a scalar multiplication is approximately

$$\left[n\mathrm{D} + \left(\tfrac{(2^{w-2}-1)(n-1)}{2^{w-2}(w+1)}\right)\mathrm{A} + \left(\tfrac{(n-1)}{2^{w-2}(w+1)}\right)\mathrm{mA}\right] \ + \ Cost_{Proposed,\ case1},$$

$$\left[n\mathrm{D} + \left(\tfrac{n-1}{w+1}\right)\mathrm{mA}\right] \ + \ Cost_{Proposed,\ case2},$$

for cases 1 and 2, respectively. Table 8 shows the performance of the scalar multiplication including the costs of the precomputation schemes proposed in this work, cases 1 and 2. As it can be seen, case 1 achieves the best performance for most common I/M ratios if $n = 160$ bits. For higher security levels ($n = 512$ bits), the difference between case 1 and case 2 reduces and, ultimately, the most effective approach would be determined by the particular I/M ratio of a given implementation. However, as the window size grows, case 1 would be again largely preferred. Therefore, for applications where memory is not scarce, case 1 would achieve the lowest cost. Similar conclusions are observed for $\mathcal{IE}$ coordinates, whose costs are not included in Table 8 because of space constraints.

Finally, we analyze the performance of the proposed scheme for tables $c_iP \pm d_iQ$. In this case, a multiple scalar multiplication using the JSF$_3$ method [9] costs approximately $[n\mathrm{D} + 0.3083\,(n-1)\mathrm{A} + 0.0617\,(n-1)\mathrm{mA}] \ + \ Cost_{Proposed,\ case1}$ and $[n\mathrm{D} + 0.37\,(n-1)\,\mathrm{mA}] \ + Cost_{Proposed,\ case2}$ for cases 1 and 2, respectively. The latter can be reduced in the case of $\mathcal{J}$ coordinates if we consider the efficient DA operation [13]. The costs in this case are expressed by $[(0.63n + 0.37)\mathrm{D} + 0.3083(n-1)\mathrm{DA} + 0.0617(n-1)\mathrm{mDA}] \ + \ Cost_{Proposed,\ case1}$ and $[(0.63n + 0.37)\mathrm{D} + 0.37(n-1)\mathrm{mDA}] \ + \ Cost_{Proposed,\ case2}$.

Table 9 shows the performance of the scalar multiplication including the costs of our precomputation scheme, cases 1 and 2. Similarly to the case of single scalar multiplication (see Table 8), case 1 achieves the best performance for most common I/M ratios for $n = 160$ bits with $\mathcal{JQ}$ and $\mathcal{IE}$ coordinates. However, if $n = 512$ bits, the range of I/M ratios for which case 2 is more efficient

Table 8. Cost of scalar multiplication using wNAF and the proposed scheme (cases 1 and 2); and I/M range for which case 1 achieves the lowest cost on $\mathcal{JQ}$ coord

Method	$n = 160$ bits		$n = 512$ bits		
	$w = 4$	$w = 5$	$w = 4$	$w = 5$	$w = 6$
Proposed, case 1	$1279.6M$	$1265.4M$	$4035.7M$	$3921.5M$	$3870.2M$
Proposed, case 2	$1I + 1267.1M$	$1I + 1269.2M$	$1I + 3970.5M$	$1I + 3874.0M$	$1I + 3858.8M$
I/M range (case 1)	$I > 12.5M$	$I > 0M$	$I > 65.2M$	$I > 47.5M$	$I > 11.4M$

Table 9. Cost of multiple scalar multiplication using JSF_3 and the proposed scheme (cases 1, 2); and I/M range in which case 1 achieves the lowest cost on $\mathcal{J}$, $\mathcal{IE}$ and $\mathcal{JQ}$

Method	$n = 160$ bits			$n = 512$ bits		
	$\mathcal{JQ}$	$\mathcal{IE}$	$\mathcal{J}$	$\mathcal{JQ}$	$\mathcal{IE}$	$\mathcal{J}$
Proposed, case 1	$1572.2M$	$1624.9M$	$1889.6M$	$4886.7M$	$5062.0M$	$5840.2M$
Proposed, case 2	$1I$+$1565.4M$	$1I$+$1635.9M$	$1I$+$1796.8M$	$1\ I$+$4771.4M$	$1I$+$4964.4M$	$1I$+$5511.1M$
I/M range (case 1)	$I > 6.8M$	$I > 0M$	$I > 92.8M$	$I > 115.3M$	$I > 97.6M$	$I > 329.1M$

increases significantly. Also, note that case 2 appears to be the best choice for $\mathcal{J}$ coordinates for a wide range of I/M ratios.

As reference, the costs for $\mathcal{JQ}$ and $\mathcal{J}$ using a traditional chain for precomputation are $1598M$ or $1I$+$1594M$, and $1922M$ or $1I$+$1849M$, respect. ($n = 160$).

6 Other Applications

We have discussed the application of the strategy of the conjugate addition to build efficient precomputation tables with the forms d_iP and $c_iP \pm d_iQ$. However, this technique can be easily applied to other table forms such as the one required by the generalized JSF [19], which requires the computation of $(3^k - 1)/2 - k$ non-trivial points. For instance, for $k = 3$ scalars, the previous algorithm requires the precomputation of $P \pm Q$, $P \pm R$, $Q \pm R$, $P + Q \pm R$, $P - Q \pm R$, which costs about 10 general additions. With our strategy, the latter is reduced to only 5 addition/conjugate addition pairs (case 1). Note that the advantage grows exponentially with the number of scalars.

Other obvious application is the extension of our strategy to other settings such as binary fields. Let us illustrate the latter with the addition formula due to [14] and later refined by [5]. The cost of adding two points $P + Q$ with the latter formula takes $13M + 4S$. Then, if we need the value $P - Q$ right after, we can store most partial results from the original addition and obtain the previous value with a cost of only $5M$ by noticing that $-Q = (X_2, X_2Z_2 + Y_2, Z_2)$ in Lopez-Dahab coordinates. Note that the partial term $Y_2Z_1^2$ from the original formula is replaced by $-Y_2Z_1^2 = (X_2Z_2 + Y_2)Z_1^2 = X_2Z_2Z_1^2 + Y_2Z_1^2$, which only cost *one* extra multiplication. Straightforward generalizations of this technique (and also of the proposed precomputation schemes) can be applied to other coordinate systems and/or elliptic curve forms.

7 Conclusions

We have introduced an innovative technique based on conjugate additions that can be efficiently exploited to reduce costs in a scalar multiplication. The relevant formulas on three different settings (namely, standard, Jacobi quartic and Edwards curves) over prime fields have been derived and shown to attain significant cost reductions in comparison with traditional formulae. In particular, we have proposed novel precomputation schemes based on this technique. Our analysis shows that the new schemes are especially attractive on the highly efficient Jacobi quartic and Edwards curves, enabling even faster implementations. Finally, we have also discussed other applications of the introduced strategy to binary fields and other precomputation tables.

Acknowledgments. We would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Centres of Excellence (OCE) for partially supporting this work. Also, after finishing this work, we became aware of a similar idea proposed by M. Scott during a presentation in ECC2008. He rediscovered the idea of exploiting similarities between $P+Q$ and $P-Q$ and suggested a slightly different sequence to precompute points.

References

1. Bernstein, D., Lange, T.: Faster Addition and Doubling on Elliptic Curves. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 29–50. Springer, Heidelberg (2007)
2. Bernstein, D., Lange, T.: Inverted Edwards Coordinates. In: Boztaş, S., Lu, H.-F(F.) (eds.) AAECC 2007. LNCS, vol. 4851, pp. 20–27. Springer, Heidelberg (2007)
3. Brown, M., Hankerson, D., Lopez, J., Menezes, A.: Software Implementation of the NIST Elliptic Curves over Prime Fields. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 250–265. Springer, Heidelberg (2001)
4. FIPS PUB 186-2: Digital Signature Standard (DSS). National Institute of Standards and Technology (NIST) (2000)
5. Higuchi, A., Takagi, N.: A Fast Addition Algorithm for Elliptic Curve Arithmetic in GF(2^n) using Projective Coordinates. Information Processing Letters 76(3), 101–103 (2000)
6. Hisil, H., Wong, K., Carter, G., Dawson, E.: Faster Group Operations on Elliptic Curves. Cryptology ePrint Archive, Report 2007/441 (2007)
7. Hisil, H., Wong, K., Carter, G., Dawson, E.: An Intersection Form for Jacobi-Quartic Curves. Personal communication (2008)
8. Koblitz, N.: Elliptic Curve Cryptosystems. Mathematics of Computation, vol. 48, pp. 203–209 (1987)
9. Kuang, B., Zhu, Y., Zhang, Y.: An Improved Algorithm for uP+vQ using JSF$_3$. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 467–478. Springer, Heidelberg (2004)
10. Lim, C.H., Hwang, H.S.: Fast Implementation of Elliptic Curve Arithmetic in GF(p^n). In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 405–421. Springer, Heidelberg (2000)
11. Longa, P.: ECC Point Arithmetic Formulae (EPAF), `http://patricklonga.bravehost.com/jacobian.html`
12. Longa, P., Miri, A.: Fast and Flexible Elliptic Curve Point Arithmetic over Prime Fields. IEEE Trans. Comp. 57(3), 289–302 (2008)
13. Longa, P., Miri, A.: New Composite Operations and Precomputation Scheme for Elliptic Curve Cryptosystems over Prime Fields. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 229–247. Springer, Heidelberg (2008)

14. López, J., Dahab, R.: Improved Algorithms for Elliptic Curve Arithmetic in GF(2^{n}). Technical Report IC-98-39, Relatorio Técnico (1998)
15. Meloni, N.: New Point Addition Formulae for ECC Applications. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007. LNCS, vol. 4547, pp. 189–201. Springer, Heidelberg (2007)
16. Miller, V.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
17. Möller, B.: Algorithms for Multi-exponentiation. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 165–180. Springer, Heidelberg (2001)
18. Okeya, K., Takagi, T., Vuillaume, C.: Efficient Representations on Koblitz Curves with Resistance to Side Channel Attacks. In: Boyd, C., González Nieto, J.M. (eds.) ACISP 2005. LNCS, vol. 3574, pp. 218–229. Springer, Heidelberg (2005)
19. Proos, J.: Joint Sparse Forms and Generating Zero Columns when Combing. Technical Report CORR 2003-23, University of Waterloo (2003)
20. Solinas, J.: Low-Weight Binary Representations for Pairs of Integers. Technical Report CORR 2001-41, University of Waterloo (2001)

A Conjugate (Mixed) Addition in Jacobian Coordinates

In the case of general addition, refer to equations (1) and (2) in Section 3.

In the case of mixed addition, let $P = (X_1, Y_1, Z_1)$ and $Q = (x_2, y_2)$ be two points on an elliptic curve E. If the mixed addition is performed using [12, formula (16)] and the partial values $(4\beta^3 + 8X_1\beta^2)$, $4X_1\beta^2$, $-8Y_1\beta^3$, Z_3 and $Z_1^3 y_2$ are temporarily stored, the conjugate mixed addition $P - Q = P + (-Q) = (X_1, Y_1, Z_1) + (x_2, -y_2) = (X_4, Y_4, Z_4)$ can be performed as follows:

$$X_4 = \gamma^2 - (4\beta^3 + 8X_1\beta^2),\ Y_4 = \gamma(4X_1\beta^2 - X_4) - 8Y_1\beta^3,\ Z_4 = Z_3 \tag{12}$$

where $\gamma = -2(Z_1^3 y_2 + Y_1)$. This formula only costs $1M + 1S + 4A + 1(\times 2)$.

B Conjugate (Mixed) Addition in $\mathcal{JQ}$ Coordinates

Let $P = (X_1, Y_1, Z_1, X_1^2, Z_1^2)$ and $Q = (X_2, Y_2, Z_2, X_1^2, Z_1^2)$ be two points on a Jacobi quartic curve. If the addition $P + Q$ is performed using the following formula due to [6]:

$$\begin{gathered} X_3 = (\alpha + 2Y_1)(\beta + 2Y_2) - \alpha\beta - 4Y_1Y_2,\ Z_3 = \phi - \theta, \\ Y_3 = (\theta + \phi + 2\alpha\beta)[4(X_1^2 + Z_1^2)(X_2^2 + Z_2^2) + a\alpha\beta + 4Y_1Y_2] - 16(X_3^2 + Z_3^2), \\ X_3^2 = (X_3)^2,\ Z_3^2 = (Z_3)^2, \end{gathered} \tag{13}$$

where $\phi = 4Z_1^2 Z_2^2$, $\theta = 4X_1^2 X_2^2$, $\alpha = (X_1 + Z_1)^2 - (X_1^2 + Z_1^2)$, $\beta = (X_2 + Z_2)^2 - (X_2^2 + Z_2^2)$, and the partial values β, $(\alpha + 2Y_1)$, $2Y_2$, $\alpha\beta$, $-4Y_1Y_2$, $(4X_1^2X_2^2 + 4Z_1^2Z_2^2)$, $2\alpha\beta$, $4(X_1^2 + Z_1^2)(X_2^2 + Z_2^2) + 4Y_1Y_2$, $a\alpha\beta$, Z_3 and Z_3^2 are temporarily stored, then the conjugate addition $P - Q = P + (-Q) = (X_1, Y_1, Z_1, X_1^2, Z_1^2) + (-X_2, Y_2, Z_2, X_2^2, Z_2^2) = (X_4, Y_4, Z_4)$ can be performed with only $2M + 1S + 7A + 1(\times 16)$ as follows:

$$\begin{gathered}X_4 = (\alpha + 2Y_1)(-\beta + 2Y_2) + \alpha\beta - 4Y_1Y_2,\ Z_4 = \phi - \theta = Z_3,\\ Y_4 = (\theta + \phi - 2\alpha\beta)[4(X_1^2 + Z_1^2)(X_2^2 + Z_2^2) - a\,\alpha\beta + 4Y_1Y_2] - 16(X_4^2 + Z_4^2),\\ X_4^2 = (X_4)^2,\ Z_4^2 = Z_3^2,\end{gathered} \tag{14}$$

In the case of mixed addition, let $P = (X_1, Y_1, Z_1, X_1^2, Z_1^2)$ and $Q = (x_2, y_2, x_2^2)$ be two points on a Jacobi quartic curve. If the mixed addition $P+Q$ is performed using the following formula due to [6]:

$$\begin{gathered}X_3 = (\alpha + 2Y_1)(x_2 + y_2) - \alpha x_2 - 2Y_1y_2,\ Z_3 = 2(Z_1^2 - X_1^2x_2^2),\\ Y_3 = 2((X_1^2x_2^2 + Z_1^2 + \alpha x_2)[2(X_1^2 + Z_1^2)(x_2^2 + 1) + a\,\alpha x_2 + 2Y_1y_2] - 2(X_3^2 + Z_3^2)),\\ X_3^2 = (X_3)^2,\ Z_3^2 = (Z_3)^2,\end{gathered} \tag{15}$$

where $\alpha = (X_1 + Z_1)^2 - (X_1^2 + Z_1^2)$, and the partial values $(\alpha + 2Y_1)$, αx_2, $-2Y_1y_2$, $(X_1^2x_2^2 + Z_1^2)$, $(2(X_1^2 + Z_1^2)(x_2^2 + 1) + 2Y_1y_2)$, $a\,\alpha x_2$, Z_3 and Z_3^2 are temporarily stored, then the conjugate mixed addition $P - Q = P + (-Q) = (X_1, Y_1, Z_1, X_1^2, Z_1^2) + (-x_2, y_2, x_2^2) = (X_4, Y_4, Z_4)$ can be performed with $2M + 1S + 7A + 2(\times 2)$ as follows:

$$\begin{gathered}X_4 = (\alpha + 2Y_1)(-x_2 + y_2) + \alpha x_2 - 2Y_1y_2,\ Z_4 = 2(Z_1^2 - X_1^2x_2^2) = Z_3,\\ Y_4 = 2((X_1^2x_2^2 + Z_1^2 - \alpha x_2)[2(X_1^2 + Z_1^2)(x_2^2 + 1) - a\,\alpha x_2 + 2Y_1y_2] - 2(X_4^2 + Z_4^2)),\\ X_4^2 = (X_4)^2,\ Z_4^2 = Z_3^2.\end{gathered} \tag{16}$$

C Conjugate (Mixed) Addition in $\mathcal{IE}$ Coordinates

Let $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ be two points on Inverted Edwards coordinates. If the general addition $P + Q$ is performed using the following formula due to [2] (note that some terms have been rearranged to save a few field additions):

$$\begin{gathered}X_3 = [\alpha + d(Z_1Z_2)^2](X_1X_2 - Y_1Y_2),\ Y_3 = [\alpha - d(Z_1Z_2)^2](X_1Y_2 + X_2Y_1),\\ Z_3 = Z_1Z_2(X_1X_2 - Y_1Y_2)(X_1Y_2 + X_2Y_1),\end{gathered} \tag{17}$$

where $\alpha = X_1X_2Y_1Y_2$, and the partial values $[X_1X_2Y_1Y_2 + d(Z_1Z_2)^2]$, X_1X_2, Y_1Y_2, $[X_1X_2Y_1Y_2 - d(Z_1Z_2)^2]$, X_1Y_2, X_2Y_1 and Z_1Z_2 are temporarily stored, then the conjugate addition $P - Q = P + (-Q) = (X_1, Y_1, Z_1) + (-X_2, Y_2, Z_2) = (X_4, Y_4, Z_4)$ can be performed with the following (with a cost of only $4M + 2A$):

$$\begin{gathered}X_4 = [\alpha - d(Z_1Z_2)^2](X_1X_2 + Y_1Y_2),\ Y_4 = -[\alpha + d(Z_1Z_2)^2](X_1Y_2 - X_2Y_1),\\ Z_4 = -Z_1Z_2(X_1X_2 + Y_1Y_2)(X_1Y_2 - X_2Y_1),\end{gathered} \tag{18}$$

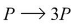
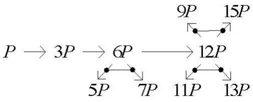
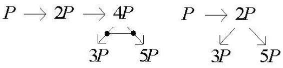
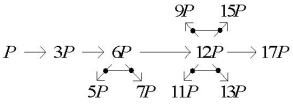
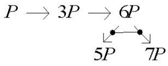
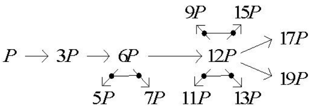
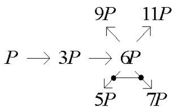
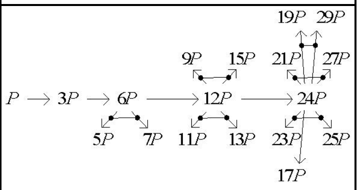
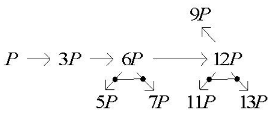
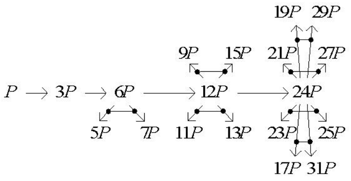
The formula for mixed addition can be obtained by setting $Z_2 = 1$ in formula (17) and has a cost of $9M + 1S + 4A$. Then, if the partial values $(X_1x_2Y_1y_2 + dZ_1^2)$, X_1x_2, Y_1y_2, $(X_1x_2Y_1y_2 - dZ_1^2)$, X_1y_2 and x_2Y_1 are temporarily cached, then the conjugate mixed addition $P - Q = P + (-Q) = (X_1, Y_1, Z_1) + (-x_2, y_2) = (X_4, Y_4, Z_4)$ can be performed by:

$$X_4 = [X_1x_2Y_1y_2 - dZ_1^2](X_1x_2 + Y_1y_2),\ Y_4 = -[X_1x_2Y_1y_2 + dZ_1^2](X_1y_2 - x_2Y_1),$$
$$Z_4 = -Z_1(X_1x_2 + Y_1y_2)(X_1y_2 - x_2Y_1), \tag{19}$$

which only costs $4M + 2A$. We remark that memory requirements of the new conjugate formulas can be minimized by performing $P + Q$ and $P - Q$ in an "interlaced" fashion (see, for instance, Table 2).

D Calculation of Precomputed Points

The table below shows the proposed precomputing sequences for various values m. For $m = 5$ the first sequence corresponds to $\mathcal{J}$ and $\mathcal{JQ}$, and the second one to $\mathcal{IE}$. Tied arrows denote addition/conjugate addition pairs (or mixed addition/conjugate mixed addition pairs if performed with affine point P).

m	Precomputation Scheme	m	Precomputation Scheme
3		15	
5		17	 9P 15P P 3P 6P 12P 17P 5P 7P 11P 13P
7		19	 9P 15P 17P P 3P 6P 12P 19P 5P 7P 11P 13P
11		29	 19P 29P 9P 15P 21P 27P P 3P 6P 12P 24P 5P 7P 11P 13P 23P 25P 17P
13		31	 19P 29P 9P 15P 21P 27P P 3P 6P 12P 24P 5P 7P 11P 13P 23P 25P 17P 31P

Practical Secure Evaluation of Semi-private Functions

Annika Paus, Ahmad-Reza Sadeghi*, and Thomas Schneider**

Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
{annika.paus,ahmad.sadeghi,thomas.schneider}@trust.rub.de

Abstract. Two-party Secure Function Evaluation (SFE) is a very useful cryptographic tool which allows two parties to evaluate a function known to both parties on their private (secret) inputs. Some applications with sophisticated privacy needs require the function to be known only to one party and kept private (hidden) from the other one. However, existing solutions for SFE of private functions (PF-SFE) deploy Universal Circuits (UC) and are still very inefficient in practice.

In this paper we bridge the gap between SFE and PF-SFE with SFE of what we call *semi-private functions* (SPF-SFE), i.e., one function out of a given class of functions is evaluated without revealing which one.

We present a general framework for SPF-SFE allowing a fine-grained trade-off and tuning between SFE and PF-SFE covering both extremes. In our framework, semi-private functions can be composed from several privately programmable blocks (PPB) which can be programmed with one function out of a class of functions. The framework allows efficient and secure embedding of constants into the resulting circuit to improve performance. To show practicability of the framework we have implemented a compiler for SPF-SFE based on the Fairplay SFE framework.

SPF-SFE is sufficient for many practically relevant privacy-preserving applications, such as privacy-preserving credit checking which can be implemented with our framework and compiler as described in the paper.

Keywords: SFE of semi-private functions, Yao's protocol, topology, optimization, compiler, privacy.

1 Introduction

Two-party Secure Function Evaluation (SFE) is an important and wide area of cryptographic research (see, e.g., [18,10,13,1,11,7,12]). It allows two parties to securely evaluate a common function on their private inputs without involving a trusted third party. The function is represented as a boolean circuit and evaluated based on a garbled version of the circuit which is created by one party (constructor Bob) and evaluated by the other party (evaluator Alice). Usually SFE hides the intermediate results but - as the function is known to both parties - not the structure (topology) of the function.

[*] The author was supported by the European Union under FP6 project SPEED.

[**] The author was supported by the European Union under FP7 project CACE.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 89–106, 2009.

In practice, however, a variety of business models require privacy properties beyond the secrecy of parties' input data to additionally keep the evaluated function private. The underlying business motivations vary from commercial incentives (e.g., protection of intellectual property) to pure security requirements to reduce the probability of credential forgery or to make insider attacks obsolete. Typical use cases are client-server applications where a user Alice inputs her private data x (hidden to Bob), the server Bob inputs his private function f (hidden to Alice), and the protocol outputs $f(x)$ to both parties such that neither party gain any information about the other party's input. Prominent examples are privacy-preserving trust negotiation schemes [3,6,4], credit checking [5], or data classification using neural networks [16].

To allow SFE of a private function, called PF-SFE [8], a universal circuit (UC) [17,8,16] is evaluated that simulates the function, and entirely hides the structure of their circuit representation. UCs require a huge overhead of $O(k \log k)$ [17], $O(k \log^2 k)$ [8], respectively $O(k^2)$ [16] additional gates, where k is the number of gates of the simulated circuit.

Fairplay [13], a state-of-the art implementation of SFE, can evaluate functions consisting of millions of gates whereas in FairplayPF [8], a recent implementation for PF-SFE, functions are restricted to a few thousand gates only due to the huge overhead for evaluating UC. Hence, a better trade-off between maximal performance (SFE) and maximal privacy of the evaluated function (PF-SFE) is desired. For many practically relevant applications (e.g., those mentioned above) it is sufficient that functions are only partly private, what we call *semi-private functions* (SPF). Basically, these applications reflect the following scenario: A user Alice has private data x, and a service provider Bob has a semi-private function $f \in \mathcal{F}$ as input, where $\mathcal{F}$ represents a given class of functions. At the end of the protocol, Alice obtains $f(x)$ but not which specific f was evaluated and Bob obtains no information on x. This problem, called *secure function evaluation of semi-private functions* (SPF-SFE), can be reduced to Yao's protocol where circuit's topology is revealed to the evaluator but the functionality of the gates is hidden. Evaluator sees the circuit topology but can only guess which functionality each part of the circuit might evaluate. We concentrate on *relaxed-security* model, i.e., security against malicious evaluator Alice and semi-honest (honest-but-curious) constructor Bob. This model is widely used in current cryptographic literature [14,2,9] and well-justified in many practical applications where performance is crucial and constructor Bob can be assumed to behave semi-honestly by means of legal contracts or possible loss of reputation.

While SPF-SFE based on Yao's protocol has been proposed as building block in many applications (e.g., [3,5,6,4,16]), we give the first unified theory for SPF-SFE. Extending and improving previously known techniques we present a general theoretical framework for SPF-SFE together with a language and tools to specify and automatically generate SPF-SFE protocols for practical applications.

Related Work. The idea of constructing circuits for a special class of functions and evaluating them efficiently with Yao's protocol in the relaxed-security model have been used in several sub-protocols [3,5,6,4,8,16]. Frikken et al. call the respective building blocks oblivious gates/circuits where evaluator does not know the function that each gate/circuit computes. However, they only mention the existence of several useful topologies like binary trees, comparison circuits, or universal circuits together with their asymptotic size, but do not give explicit constructions. We extend their basic ideas into a generic framework and provide a wide class of functional blocks, each with a concrete efficient implementation (topology, programming, and exact size), that can be arbitrarily combined to represent semi-private functions in many practical applications.

Existing frameworks for secure computation based on Yao's protocol are the Fairplay SFE system [13] with a proposed extension to the malicious model [12] and another extension to private functions with UCs (PF-SFE), called FairplayPF [8]. The Fairplay compiler includes an optimizer that optimizes on the basis of the high-level Secure Function Description Language (SFDL) using peekhole optimization, duplicate code removal, and dead code elimination. In contrast to this, our proposed optimization algorithm for constant inputs optimizes on the lower abstraction level of circuits and can also be applied to further optimize the output of circuits generated with the Fairplay compiler.

Our Contribution and Outline. We propose a general framework together with a compiler for efficient secure function evaluation of semi-private functions (SPF-SFE) in the relaxed-security model.

In §2 we describe how common SFE can be extended with building blocks that we call *Privately Programmable Blocks* (PPB) to allow practical secure evaluation of semi-private functions (SPF-SFE). A privately programmable block (§4) consists of a fixed topology of several programmable gates (with a small number of inputs) and can be programmed to evaluate different functions out of a class of functions. The evaluator learns how the blocks are connected (topology) but *not* with which of the functions of their corresponding class of functions the blocks are programmed. Hence parts of the function are hidden from the evaluator while the topology is still revealed. In §5 we show how to design efficient constructions for PPBs that also allow to securely incorporate private constants into PPBs and give concrete constructions that are of special interest for practical applications. In particular we present efficient PPB constructions to compare two numbers and a number with a private constant. Other efficient PPB constructions for arithmetic operations (add or subtract two numbers/a number and a private constant, multiply a number with a private constant) and boolean operations are given in the full version of this paper [15]. Also switching functions, e.g., permutation and selection blocks, as well as universal circuits from [8] fit into this concept. The resulting SPF-SFE protocol is as efficient as Yao's SFE protocol while providing function privacy at the same time.

In §8 we present an optimization algorithm that incorporates constant inputs into the circuit resulting in a circuit with less inputs and smaller size having a topology which is *independent* of the values of the constant inputs. Besides the well known propagation of constant inputs, our algorithm additionally eliminates resulting gates with one input by incorporating them into surrounding gates which results in smaller circuit size. The proposed optimization algorithm applies no cryptographic modification of circuits and hence is of independent interest. This optimization can be used in combination with Yao's SFE protocol in the relaxed-security scenario where constant inputs might be public values known to both parties as well as the inputs of circuit constructor Bob.

In order to allow usage of SPF-SFE in many practical applications we present a general compiler framework for secure evaluation of semi-private functions, called FairplaySPF, based on the well known Fairplay SFE system [13] as described in §6. Our Secure Programmable Block Description Language (SPBDL) allows to specify the topology of interconnected programmable blocks together with their corresponding private programming. A compiler automatically compiles SPBDL descriptions to circuits described in Fairplay's Secure Hardware Description Language (SHDL). After incorporating Bob's inputs into the circuit with the optimization algorithm presented in §8, the circuit can securely be evaluated with the SPF-SFE protocol while hiding the programming. Also a Universal Circuit (UC) that is evaluated in PF-SFE (cf. [8]) can be seen as a PPB that is programmed with a private circuit (specified in SHDL). By incorporating UCs as programmable blocks into SPBDL, our framework becomes a general purpose framework capable of expressing SFE, SPF-SFE, and PF-SFE as well as arbitrary combinations of them where only sensitive parts of the function's structure are hidden as shown in the example in §7. This allows a fine-grained trade-off between performance and privacy of the evaluated function.

Our framework and compiler can be applied (combining SPF-SFE and PF-SFE) to implement and improve efficiency of several applications such as privacy-preserving credit checking [5], blinded policy evaluation [3,6,4], or secure data classification [16]. In §7 of this paper we concentrate on privacy-preserving credit checking. Usually, before getting a loan from a bank a person has to reveal a substantial amount of private information. This information has to satisfy certain criteria that are defined by the bank. We show how SPF-SFE can be used to securely evaluate the trustworthiness of a borrower while ensuring that (i) the privacy of his input is preserved and (ii) nothing is revealed about the criteria of the bank used for credit checking. Instead of using a UC for the whole function as in PF-SFE we reveal the topology of the trivial part of the function (e.g., comparing attributes with thresholds) and only hide the sensitive part in a UC, which is much more efficient. The description of the function in SPBDL can automatically be compiled into SHDL code with our compiler. This can be obliviously evaluated in a one-round protocol.

2 Yao's Protocol and Semi-private Functions

Yao's Protocol. In the following, we concentrate on Yao's protocol [18] for SFE. Yao's protocol is often called *garbled circuit* protocol as a garbled version of the (boolean) circuit representing the function is created by one party (constructor Bob) and evaluated by the other party (evaluator Alice) as described in the following. For each wire of the circuit, Bob uses two random bit strings (garbled values) that are assigned to the corresponding values 0 and 1, respectively. Note, that the garbled values do not reveal to which value they correspond as they are chosen randomly. Bob sends *only* the garbled values corresponding to his inputs (garbled inputs) to Alice. For Alice's inputs, Bob uses 1-out-of-2 oblivious transfer (OT) to send Alice *only* the garbled values corresponding to her inputs without Bob learning which strings she gets. Additionally, for each gate G_i of the circuit, Bob creates and sends to Alice a *garbled table* T_i with the following property: given garbled values for G_i's inputs, T_i allows to recover *only* the garbled value of the corresponding output of G_i and nothing else. Afterwards, Alice uses the received garbled values of the input wires and garbled tables T_i to evaluate the garbled circuit gate by gate. The output wires of the circuit are not garbled (or the mappings from garbled values to values 0 and 1 are published by Bob), thus Alice learns (only) the output of the circuit, but no plain values of internal wires (only garbled values). Correctness and security against semi-honest adversaries of Yao's protocol are proven in [10]. It is easy to show that Yao's protocol is even secure against malicious Alice, i.e. relaxed-secure, as the only message Alice sends to Bob is within the OT protocol where Alice is unable to cheat successfully if the OT protocol is secure against malicious Alice [4, Appendix A]. An efficient OT protocol with relaxed-security is given in [2].

Yao's protocol is the kernel of existing implementations of SFE protocols [13,12] which also extend it to be secure against malicious constructor Bob via cut-and-choose, e.g., multiple circuits are garbled, correctness of some of them is verified by revealing all garbled input values (called open), and the remaining ones are evaluated. As justified in the introduction, we concentrate on the plain Yao's protocol (secure against semi-honest Bob and potentially malicious Alice) where only *one* circuit is evaluated and no circuits are opened.

Yao's Protocol for Semi-Private Functions. Observe, in Yao's protocol the garbled tables T_i consist of symmetric encryptions of the garbled output value using the corresponding garbled input values as keys. Alice can use these garbled input values to decrypt exactly the one garbled output value corresponding to these keys. All other garbled output values, i.e., entries of the garbled function table remain hidden from Alice and hence she cannot determine the type of the gate. The only information Alice learns about the function in Yao's protocol is the *topology* of the circuit, i.e., the way the different gates are connected and how many inputs each gate has.

When Alice obtains a garbled circuit from Bob, she can guess from its topology what functionality the circuit evaluates, e.g., chains of 3-input gates might be an integer comparison circuit. This can be exploited constructively by Bob to keep parts of the function private, we call this a *semi-private function*, as follows. Bob composes his intended functionality from blocks with a fixed topology that can evaluate different functionalities each, called *privately programmable blocks* (PPBs) as explained in §4. The maximum amount of information Alice can gain from the topology of a PPB is the set of functionalities the PPB might compute but not the specific functionality chosen privately by Bob.

From combining these two arguments follows that evaluation of a circuit, composed out of several PPBs representing the semi-private function, with Yao's protocol is a secure protocol for SPF-SFE.

Additionally, (semi-honest) Bob can incorporate his input values into the circuit before garbling the circuit if they are already known at that time. In §8 we give an algorithm for efficient optimization of circuits for Bob's (constant) inputs together with an example. The optimization *only* depends on the topology of the original circuit but not on Bob's input values and hence the optimized circuit does not reveal more information on Bob's input values than the original circuit. After this optimization, Bob no longer needs to transfer the garbled values corresponding to his input values to Alice and also the size of the circuit is reduced (resulting in less communication and computation).

3 Definitions and Preliminaries

Let $x \in [0, 2^\ell)$ be an unsigned ℓ-bit integer value and $\boldsymbol{x} = (x_1, .., x_\ell)$, $x_i \in \{0, 1\}$ its corresponding representation as bit vector, i.e., $x = \sum_{i=1}^{\ell} x_i 2^{i-1}$. The *length* of $\boldsymbol{x}$ is $|\boldsymbol{x}| = \ell$. We draw a (single) *wire* with one-bit value as $\longrightarrow$. As usual, *multi wire* X with ℓ-bit value x is drawn as $\overset{\ell}{\not\longrightarrow}$ and consists of ℓ wires indexed by $X[i]$, $i = 1, .., \ell$ with values x_i.

A *gate* G with degree d has d *inputs* and one *output*. It is the implementation of a boolean function $g : \{0, 1\}^d \rightarrow \{0, 1\}$. As special case, a *constant gate* has no inputs ($d = 0$) and outputs a constant value. The *size* of a gate G, denoted by $|G|$, is the number of function table entries needed to implement the gate, namely $|G| = 2^d$. A gate with $e > 0$ outputs can easily be combined from e gates with one output resulting in size $e \cdot 2^d$.

We consider acyclic *circuits* consisting of connected gates with arbitrary fan-out, i.e., the output of each gate can be used as input to arbitrary many gates. The *size* of a circuit, denoted by $|C|$, is the sum of the sizes of its gates. Note, communication and computation complexity of efficient SFE protocols is linear in the size of the circuit.

A *block* B_v^u is a sub-circuit with u inputs $in_1, .., in_u$ and v outputs $out_1, .., out_v$. B_v^u computes function $f_B : \{0, 1\}^u \rightarrow \{0, 1\}^v$ mapping input values to output values. Blocks consist of connected gates and other sub-blocks. *Size* of block B, denoted by $|B|$, is the sum of the sizes of its sub-elements.

A *programmable gate* (PG) is a gate with an unspecified function table. *Programming* it is done by providing a specific function table with 2^d entries (one entry for each input combination). The concept of PGs corresponds to a universal circuit for simulating a single gate in Valiant's UC construction [17]. As described in the previous section, in SPF-SFE evaluator Alice is not able to extract the corresponding function table (program) from PG. Analogously, a *programmable block* (PB) is a block consisting of programmable gates or programmable sub-blocks. It is programmed by programming each of its sub-elements. As described before, in SPF-SFE evaluator Alice is unable to extract the program from PB.

4 Privately Programmable Blocks

In this section we present the concept of *Privately Programmable Blocks* (PPB) for constructing semi-private functions. Using our efficient PPB constructions given in §5 with the SPF-SFE protocol of §2 allows to preserve the privacy of the function while the protocol remains as efficient as SFE protocol.

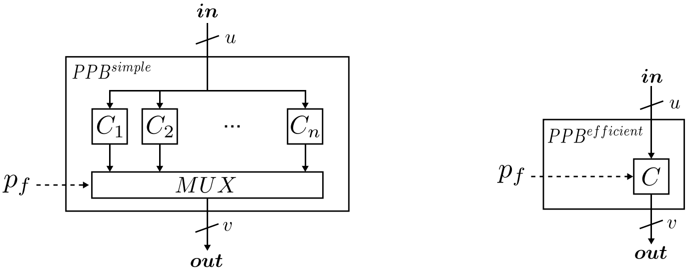
Definition 1. *A* Privately Programmable Block *(PPB) is a programmable block which can be programmed to compute any function f of a given class of functions $\mathcal{F}$ (e.g., $\mathcal{F} = \{ADD, SUB\}$) with a corresponding program p_f (e.g., $f = ADD$). We write PPB^f for a PPB which is programmed to compute f:*

$$\forall f \in \mathcal{F}, \forall (in_1, .., in_u) \in \{0,1\}^u : PPB^f(in_1, .., in_u) = f(in_1, .., in_u).$$

As explained in §2 before, in SPF-SFE the function to be evaluated is composed of several PPBs. Evaluator Alice learns how the PPBs are connected (topology), but the programming of the PPBs remains to be private information of constructor Bob (that's why PPBs are called *privately* programmable). Alice can infer from the topology of a PPB at most the class of possible functionalities $\mathcal{F}$ but not the specific functionality f chosen by Bob. Hence, from Alice's point of view the PPB can compute any functionality from $\mathcal{F}$ and the amount of information hidden inside the PPB is $\log_2 |\mathcal{F}|$ bits. For a semi-private function which is composed from programmable blocks $PPB_1, .., PPB_n$, the program of each PPB can be combined with any programming of the other PPBs and hence the maximum (as some combinations might not make sense depending on the application) amount of information hidden in the semi-private function is $\log_2(|\mathcal{F}_1| \cdot .. \cdot |\mathcal{F}_n|) = \sum_{i=1}^{n} \log_2 |\mathcal{F}_i|$ bits. Clearly, if this is not large enough (i.e., if the number of PPBs n or number of possible functionalities of PPBs $|\mathcal{F}_i|$ is small), Alice might guess the correct function with high probability or probe the system via exhaustive search which must be prohibited by other means.

Universal Circuits (UC) indeed are special PPBs that can be programmed to compute an arbitrary function. UC_k is capable of simulating *any* function corresponding to a circuit with up to k gates with two inputs each. UCs provide *full privacy* of the evaluated function as the topology is hidden entirely. However, they cause a huge overhead by increasing the size of the evaluated circuit by $O(k \log k)$ [17], $O(k \log^2 k)$ [8], or $O(k^2)$ [16] additional gates which is often intolerable in practice. Evaluating a UC programmed with a private function known by constructor Bob with a SFE protocol is called Secure Evaluation of Private Functions (PF-SFE). By combining the PPBs presented in this paper with UCs, users can find a fine-grained trade-off between efficient PPB constructions for semi-private functions (SPF-SFE) and less efficient UC constructions for completely private functions (PF-SFE) as explained in §7.

Simple PPB Construction. A straight-forward implementation of a PPB for a class of n arbitrary functionalities $\mathcal{F} = \{f_1, f_2, .., f_n\}$ can directly be derived from the definition of PPBs in Definition 1 as shown in Fig. 1(a). Each functionality f_i is computed by a circuit C_i and a $n:1$ multiplexer (MUX) is



(a) Simple PPB construction (b) Efficient PPB construction

Fig. 1. PPB constructions

programmed to select the intended output. The MUX block can be constructed from v parallel selection blocks S_1^n (as defined in [8]) for each of the v outputs that can be programmed to select any of their n inputs as outputs.

If the program p_f is known by Bob beforehand it can directly be incorporated into the circuit as described in §8. After optimization, each of the v selection blocks consists of a chain of $n-1$ programmable 2-input gates which can be programmed to select either their left or right input as output each [8]. The size of this simple PPB construction is $\left|PPB^{simple}\right| = 4v(n-1) + \sum_{i=1}^{n} |C_i|$.

Efficient PPB Constructions. *Efficient PPB constructions* can be obtained by choosing special classes of functionalities having circuits with the same (or at least a similar) topology. This allows to re-use (parts of) the same circuit C for the different functionalities f_i as shown in Fig. 1(b). For instance, the topology of an adder is the same as that of a subtractor and hence for $\mathcal{F} = \{ADD, SUB\}$ the same topology can be used. Based on the intended functionality $f \in \mathcal{F}$, the sub-elements of C are programmed differently while the topology is the same. This efficient PPB construction has size $\left|PPB^{\text{efficient}}\right| = |C| \approx |C_i| \ll \left|PPB^{simple}\right|$.

When a private constant $\boldsymbol{c}$ is incorporated into a PPB, the *value of the constant can not be extracted from PPB's topology* and hence is hidden from Alice in the SPF-SFE protocol, e.g., circuits to add/subtract an input with a s-bit constant $\boldsymbol{c}$ have the same topology. To simplify notation, we parametrize the class of possible functionalities with parameter c and write $\mathcal{F}_c = \{f_{1_c}, .., f_{n_c}\}$ for $\mathcal{F} = \{f_1|_{c=0}, .., f_1|_{c=2^s-1},\ f_2|_{c=0}, .., f_2|_{c=2^s-1}, \ldots, f_n|_{c=0}, .., f_n|_{c=2^s-1}\}$, e.g., $\mathcal{F}_c = \{ADD_c, SUB_c\}$ in the example given above. The amount of information hidden inside a PPB is

$$log_2 |\mathcal{F}| = log_2 |\mathcal{F}_c| + |c| = \log_2(n) + s \text{ bits}. \qquad (1)$$

5 Practical Efficient PPB Constructions

In this section we show how to construct several efficient PPBs that are useful in practical applications (cf. §7). All these building blocks are already implemented in our framework for practical SPF-SFE described in §6. In the following we present two efficient PPB constructions for *arithmetic operations*: compare two numbers and a number with a private constant. Other efficient PPB constructions for arithmetic operations (add or subtract two numbers/a number and a private constant, multiply a number with a private constant) and boolean operations are given in the full version of this paper [15]. Our SPF-SFE framework also provides PPBs for *Switching Functions* (i.e., permutation and selection blocks) and *Universal Circuits* for which we refer to the definitions, descriptions, and constructions in [8]. A list of efficient PPB constructions provided for implementation in our framework is given in the full version of this paper [15].

For each PPB we give the *Interface* specifying the functionality of the block, its number of inputs and outputs, and the different possibilities for programming $\mathcal{F}_c$. The *Implementation* describes the topology of the corresponding efficient PPB construction, how to program it, and its size. The inputs are called $\boldsymbol{x}$, $\boldsymbol{y}$ and the potential private constant is called $\boldsymbol{c}$, where $|\boldsymbol{x}| = m$, $|\boldsymbol{y}| = n$, and $|\boldsymbol{c}| = s$. To simplify presentation we assume w.l.o.g. $m = n$, respectively $m = s$ in the following descriptions. The other cases can easily be derived from these by padding the shorter input with zeros and optimizing constant inputs afterwards as described in §8. Recall, evaluator Alice can neither extract the chosen function $f_c \in \mathcal{F}_c$, nor the value of the possibly embedded private constant $c \in \{0,1\}^s$, from the topology of any PPB. The amount of information hidden inside the PPB is given by equation (1).

The main idea underlying efficient PPB constructions is to combine functionalities that have structurally equivalent recursive definitions that directly translate into programmable gates of equivalent topologies. E.g., comparison if two m-bit numbers $\boldsymbol{x}, \boldsymbol{y}$ of bitlength m are less or equal is defined recursively as

$$(x \leq y) \Leftrightarrow \Big((x_m < y_m) \vee \big((x_m = y_m) \wedge ((x_{m-1}, .., x_1) \leq (y_{m-1}, .., y_1))\big)\Big). \quad (2)$$

Whether two numbers are greater or equal is defined recursively as

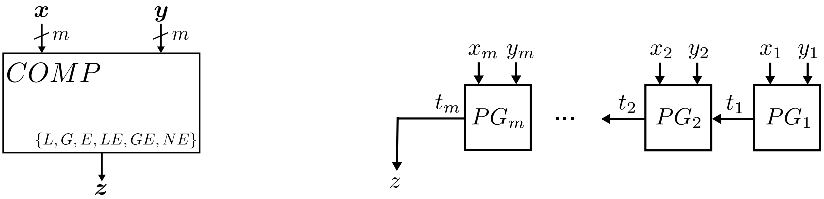
$$(x \geq y) \Leftrightarrow \Big((x_m > y_m) \vee \big((x_m = y_m) \wedge ((x_{m-1}, .., x_1) \geq (y_{m-1}, .., y_1))\big) \quad (3)$$

which is structurally equivalent and translates into the same topology (Fig. 2(b)).

5.1 PPB:COMP - Compare Two Numbers

Interface (Fig. 2(a)). PPB_{COMP} implements $z = f(x,y) = x \bowtie y$, where $\bowtie \in \{<, >, =, \leq, \geq, \neq\}$ and $|\boldsymbol{z}| = 1$. The corresponding class of functions is $\mathcal{F} = \{L, G, E, LE, GE, NE\}$.

Implementation (Fig. 2(b)). Topology of PPB_{COMP} consists of a chain of m programmable gates PG_i (full comparers) with input bits x_i, y_i, and carry-in t_{i-1} and output carry-out t_i. The output of PPB_{COMP} is $z = t_m$ and the first carry $t_0 = 1$ can be directly incorporated into PG_1. The carry t_i propagates whether for the i least significant bits $x_{<i} = x \; mod \; 2^i$ and $y_{<i} = y \; mod \; 2^i$ the corresponding relation is fulfilled ($t_i = 1$) or not ($t_i = 0$). In the following we describe the



(a) Interface (b) Topology

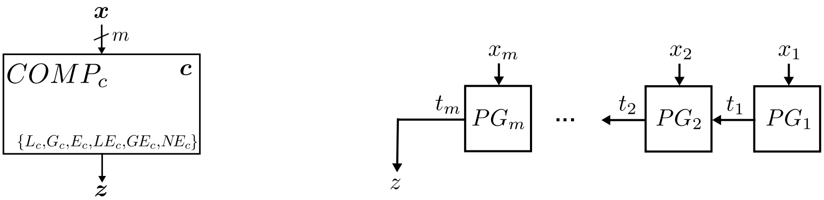
Fig. 2. PPB:COMP

programming for the cases $=$, $\leq$, and $\geq$; the corresponding cases $\neq$, $>$, and $<$ can be easily derived from this by negating output t_m in PG_m. In case $f = E$, PG_i is programmed to compute $t_i = (x_i = y_i) \wedge (x_{<i} = y_{<i}) = (x_i = y_i) \wedge t_{i-1}$. Analogously, in case $f = LE$, PG_i computes $t_i = (x_i < y_i) \vee [(x_i = y_i) \wedge t_{i-1}]$ and in case $f = GE$, PG_i computes $t_i = (x_i > y_i) \vee [(x_i = y_i) \wedge t_{i-1}]$. Note, these function table entries correspond exactly to the recursive definitions in equation (2) and (3). This block has size $|PPB_{COMP}| = (m-1) \cdot 2^3 + 2^2 = 8m - 4$.

5.2 PPB:COMPc - Compare Number with Private Constant

Interface (Fig. 3(a)). PPB_{COMPc} implements $z = f_c(x) = x \bowtie c$, where $\bowtie \in \{<, >, =, \leq, \geq, \neq\}$, $\boldsymbol{c}$ is a private constant hidden inside PPB, and $|\boldsymbol{z}| = 1$. The corresponding class of functions is $\mathcal{F}_c = \{L_c, G_c, E_c, LE_c, GE_c, NE_c\}$.

Implementation (Fig. 3(b)). Topology of PPB_{COMPc} is exactly the same as that of PPB_{COMP} described in the previous section, however, each programmable gate PG_i has no input for y_i which is replaced by the internal constant c_i. The programming is the same as for PPB_{COMP} with constant c_i instead of input y_i. This block has size $|PPB_{COMPc}| = (m-1) \cdot 2^2 + 2^1 = 4m - 2$.



(a) Interface (b) Topology

Fig. 3. PPB:COMPc

6 FairplaySPF - A General Framework for SPF-SFE

We have implemented a general framework for Secure Evaluation of Semi-Private Functions (SPF-SFE) called *FairplaySPF* [1] by extending the Fairplay SFE

[1] FairplaySPF is available for download at `http://www.trust.rub.de/FairplaySPF`.

framework [13], both written in JAVA. Fairplay provides two languages: The high-level Secure Function Description Language (SFDL) allows users to specify the functionality to be computed with elements known from other high-level hardware description languages like VHDL or Verilog (e.g., variables, arrays, procedures, arithmetic- and logic expressions, control structures, etc.). Fairplay optimizes the function described in SFDL and automatically transforms it into a boolean circuit described in Fairplay's low-level Secure Hardware Description Language (SHDL). This language consists of wires, input wires, gates, and output gates only. Using the SHDL circuit as input for both parties, Alice and Bob invoke their respective programs of the Fairplay runtime environment to execute the two-party SFE protocol. These programs evaluate the function on their respective private inputs over a TCP connection.

FairplaySPF Framework. In FairplaySPF, we extend the Fairplay framework [13] to secure evaluation of semi-private functions that are known to Bob only. In the following we describe the workflow of the FairplaySPF framework. Bob composes his semi-private function from several available privately programmable blocks (as described in §5) in our newly designed *Secure Programmable Block Description Language (SPBDL)* explained later in this section. Our FairplaySPF compiler automatically translates this SPBDL program into an SHDL circuit. Alternatively, SHDL circuits that are generated by the original Fairplay compiler from SFDL descriptions can be used. Bob's private input data is automatically incorporated into the SHDL circuit and optimized afterwards by the FairplaySPF circuit optimizer as described in §8 resulting in a smaller SHDL circuit. This optimized SHDL circuit (containing the combination of Bob's semi-private function and his private data) is evaluated by the FairplaySPF runtime environment (RE) which is only a slight modification of the Fairplay RE for semi-private functions: In FairplaySPF RE *only Bob* inputs the SHDL circuit but not Alice. The topology of the circuit (but without the types of the gates) is sent to Alice and afterwards the SPF-SFE protocol as described in §2 is executed between Alice and Bob over a TCP connection.

Secure Programmable Block Description Language (SPBDL). Our new SPBDL language allows to easily specify semi-private functions by combining different PPBs. SPBDL extends the basic functionality of SHDL to input wires (`input`), multi-wires (`vector`), privately programmable blocks (`block`), programmable gates (`gate`), and output wires (`output`). The formal specification of the *syntax* of SPBDL in Extended Backus-Naur Form (EBNF) is given in the full version of this paper [15]. In the following, we briefly describe the *semantics* of SPBDL. Please see Fig. 4 for an example SPBDL description of a semi-private function. As in SHDL, each line of a SPBDL program starts with a line number beginning with 0. In following lines, this number refers to the output of the element defined in this line. Line comments start with `//`.

A SPBDL program starts with the definition of inputs as `input Player [w]`, where `Player` defines from which party the input is given (`alice` or `bob`). The optional parameter `[w]` specifies that the input consists of w bits (default $w = 1$).

Afterwards, three kinds of elements can be specified - `gate`, `vector`, and `block`: A programmable gate is defined as `gate in [Wires] p [Bits]`, where `Wires` is its list of inputs and `Bits` is the programming of its function table. A list of `Wires` can be grouped into a vector with `vector [Wires]`. The single wires of a vector can be accessed via `Vector.Index`, e.g., `4.2` denotes the second wire of vector 4. A PPB is defined as `block [Btype] out Num in [Vects] p [Bprog]`, where `Btype` is the type of the PPB (e.g., `comp` for PPB_{COMP} described in §5), `Num` specifies the number of output bits, and `Vects` is the list of input vectors. The programming of the PPB specified in `Bprog` depends on the type of the PPB `Btype`. All types of PPBs `Btype` and corresponding programming parameters `Bprog` available in SPBDL are given in the full version of this paper [15]. Finally, outputs are defined as `output Player Vect`, where `Player` defines which party obtains the output (`alice` or `bob`) and `Vect` is the vector to be output.

7 Applications

Our general framework and tools for SPF-SFE presented in this paper can be used to specify and implement many privacy-preserving applications. Examples are *Blinded Policy Evaluation* [3,6,4], *Privacy-Preserving Credit Checking* [5], or provably secure evaluation of *Private Neural Networks* [16].

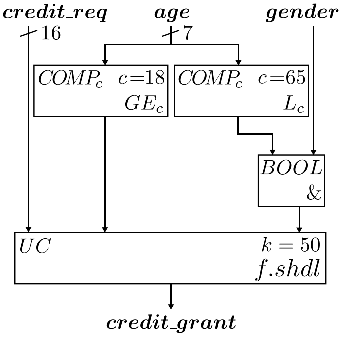
In the following we concentrate on privacy-preserving credit checking [5] which demonstrates how the evaluated function can be partitioned into semi-private and private parts which are both supported by our framework.

Privacy-Preserving Credit Checking. Typically, before granting a loan from a lender (Bob), the credit worthiness of the borrower (Alice) is checked to have the confidence that she will be able to pay it back later. The borrower is asked for her credit report that contains a large amount of private information including for example gender, age, income, salary, or other sensitive information like how many trade lines she owns, the number of overdrafts, or the number of late payments. However, revealing this data should be avoided as the lender may not always be a credible organization or, even worse, dishonest employees (so called insiders) could sell such private information on customers to third parties.

Additionally, the evaluation criteria of the lender are highly sensitive information that must be protected as revelation of these may cause loss of intellectually property or loss of repudiation for the lender.

As suggested by Frikken et al. [5], this scenario can be reduced to SPF-SFE, where Alice inputs her private credit report and Bob evaluates his semi-private function that checks if the credit report fulfills his criteria. To ensure that Alice inputs correct data into the SPF-SFE protocol, the authors describe how to replace the oblivious transfer step by a Credit Report Agency, i.e., a trusted third party, that checks and accredits Alice's inputs instead.

Bob's semi-private credit checking function can be expressed in our framework for SPF-SFE as shown in the tiny example of Fig. 4 which is due to space limitations not intended to give the complete solution but merely to show the



```
0 input alice [7]  // age
1 input alice      // gender
2 input alice [16] // credit_req
3 block [compc] out 1 in [0] p [GE 18]
4 block [compc] out 1 in [0] p [L 65]
5 block [bool] out 1 in [4 1] p [AND]
6 vector [2 3 5]
7 block [uc] out 1 in [6] p [50 f.shdl]
8 output alice 7 // credit_grant
9 output bob   7 // credit_grant
```

Fig. 4. Example for Privacy-Preserving Credit Checking

main concepts. The upper part of the circuit performs some obvious computation on Alice's data, e.g., compare her *age* with a private constant, or combine this result with her *gender*. The sensitive information in this part of the function are the private constants, e.g., grant credit only to female persons ($gender = 1$) that are younger than 65 ($age < 65$), which are hidden from Alice, whereas the obvious topology can safely be revealed.

The highly sensitive part of the functionality that combines these results depending on the amount of credit requested (*credit_req*) is hidden entirely from Alice within the universal circuit *UC*. This PPB can be programmed to compute *any* functionality computable by a circuit of up to $k = 50$ gates with arbitrary topology. The specific functionality intended by Bob is the SHDL circuit described in `f.shdl`, which can automatically be generated from a high-level description in SFDL with the Fairplay compiler.

This example shows how our framework for SPF-SFE can be used to implement an application-specific, reasonable tradeoff between efficiency while revealing irrelevant information (SPF-SFE with PPBs) and complete function privacy (PF-SFE with UC).

Comparison of SPF-SFE and PF-SFE. Revealing the topology of obvious parts of the functionality while hiding the sensitive parts in a UC results in a smaller circuit as UC overhead can be substantially reduced due to less simulated gates k and less inputs into UC. This reduced size of the evaluated circuit directly translates into corresponding speedups in *any implementation* of the underlying SPF-SFE protocol as their performance must be at least linear in the size of the evaluated circuit.

As concrete example, Table 1 shows the number of gates that can be saved in the privacy-preserving credit checking example of Fig. 4 compared to hiding the functionality entirely in a UC in PF-SFE. For different maximum size k (row A) of the part of the functionality which is hidden in UC we give the achieved performance improvements when extracting the obvious part of the functionality into the upper part of the circuit ($COMP_c$ blocks and $BOOL$ block in Fig. 4). In our example, these blocks consist of 14 gates, i.e., row B contains the fraction

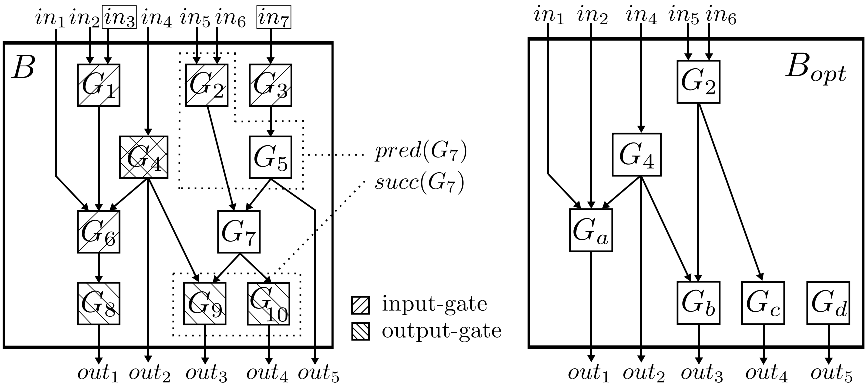
Table 1. Improved UC Overhead in the Example of Fig. 4

A) Gates hidden in UC, k	25	50	100
B) Gates extracted, $14/(k+14)$	35.9%	21.9%	12.3%
C) UC overhead in PF-SFE (UC type)	$1,861$ (M3)	$3,720$ (M3)	$8,264$ (M3)
D) UC overhead in SPF-SFE (UC type)	850 (M1)	$2,571$ (M3)	$6,797$ (M3)
E) Improvement SPF-SFE vs. PF-SFE	$1,011$ (54.3%)	$1,149$ (30.9%)	$1,467$ (17.8%)

of the functionality which is revealed: $14/(k+14)$. Row C shows how many gates are needed to hide the whole functionality of $14+k$ gates in a UC with 24 inputs (for *credit_req*, *age*, and *gender*) using the most efficient UC construction of [16] which is denoted in parentheses. Row D shows how many gates are needed to implement the UC in our mixed approach as shown in Fig. 4, where UC has 18 inputs and simulates k gates. The resulting improvements compared to the PF-SFE solution (row E) supersede the fraction of the gates extracted (row B) as the number of inputs into UC is also reduced.

8 Optimization of Circuits with Constant Inputs

We describe a general optimization algorithm that incorporates constant inputs into a block (sub-circuit) B. The topology of the resulting optimized block B_{opt} is *independent* of the values of the constant inputs and its number of inputs and size are smaller, i.e., the number of gates respectively their degree is reduced as shown in Fig. 5. Besides the well known propagation of constant inputs (step 1), our algorithm additionally eliminates resulting gates with one input by incorporating them into surrounding gates (steps 2 and 3), which results in a smaller circuit size. The optimization algorithm is a non-cryptographic transformation of circuits and hence of independent interest. As outlined in §2, one possible


in_1 in_2 in_3 in_4 in_5 in_6 in_7
B
G_1
G_2
G_3
G_4
G_5
G_6
G_7
G_8
G_9
G_{10}
$pred(G_7)$
$succ(G_7)$
▨ input-gate
▧ output-gate
out_1 out_2 out_3 out_4 out_5
in_1 in_2 in_4 in_5 in_6
B_{opt}
G_2
G_4
G_a
G_b
G_c
G_d
out_1 out_2 out_3 out_4 out_5

(a) Block with constant inputs in_3 and in_7 (b) Block after optimization

Fig. 5. Example for circuit optimization with Algorithm 1

application is to use this optimization to improve Yao's protocol. In this application, constant inputs might be public constant values known to both parties as well as the private inputs of (semi-honest) circuit constructor Bob (if known at the time of construction of the garbled circuit).

Terminology. The following terminology is visualized in Fig. 5(a). Assume the gates G_i, $i = 1, .., n$ of a block B are numbered in topological order, i.e., gate G_i has no inputs that are outputs of gates with larger index $G_{j>i}$. Otherwise, this order can be obtained efficiently via topological sorting in $O(n)$.

An *output gate* is a gate whose output is also an output of B. Similarly, an *input gate* is a gate, which has at least one input that is also an input of B. For gate G_i, $pred(G_i)$ denotes the set of its predecessors, i.e., gates whose output is an input into G_i. Analogously, $succ(G_i)$ denotes the set of G_i's successors, i.e., gates having the output of G_i as input. The fan-out of a gate G_i is the number of its successors, i.e., $fanout(G_i) = \#succ(G_i)$.

Optimization. We refer to the running example of Fig. 5 that optimizes a block B with constant inputs in_3 and in_7 in the following description of Algorithm 1.

Step 1 - Eliminate constant inputs. The first step of Algorithm 1 eliminates all constant inputs c_j, $j = 1, .., c$ of block B with respective constant value $v_j \in \{0, 1\}$. For all gates G_i with degree d_i having c_j as k_i-th input, the function **eliminateConstInput**(G_i, k_i, v_j) is called that eliminates the corresponding input of G_i. Only the lines of the function table of G_i with value v_j in the k_i-th position are used while the other entries are eliminated, i.e., the modified gate $G_{i'}$ computes $g_{i'}(in_1, .., in_{k_i-1}, in_{k_i+1}, .., in_{d_i}) = g_i(in_1, .., in_{k_i-1}, v_j, in_{k_i+1}, .., in_{d_i})$.

Algorithm 1. Optimize block B with constant inputs

```
Input: Block $B$ of gates $G_1, .., G_n$ in topological order
Output: Optimized block $B_{opt}$
begin
1   # Eliminate constant inputs
    forall constant inputs $c_j$ with constant value $v_j$ that are not outputs of $B$ do
        forall gates $G_i$ having $c_j$ as $k_i$-th input do
            eliminateConstInput($G_i$, $k_i$, $v_j$)
2   # Eliminate non-output gates with one input
    forall non-output gates $G_i$ with $d_i = 1$ do
        integrateInSucc($G_i$)
3   # Eliminate output gates with one input
    forall output gates $G_i$ with $d_i = 1$ do
        let $\{G_p\} = pred(G_i)$
        if $G_i$ is not input gate and $fanout(G_p) = 1$ then
            integrateInPred($G_i$,$G_p$)
end
```

$|G_i|$ shrinks by a factor of two for each of its constant inputs. Let $\#c_i$ denote the number of constants of the d_i inputs of G_i, then $|G_{i'}| = 2^{d_i - \#c_i}$ after Step 1 of Algorithm 1 has eliminated all constant inputs. For an efficient implementation of Algorithm 1 it is crucial that **eliminateConstInput()** does not copy the entire function table of a gate G_i for each elimination of a constant input as this would result in runtime $O(\#c_i \cdot |G_i|)$ for each gate. Instead, the constant inputs are marked in runtime $O(\#c_i)$ and afterwards all constant inputs are eliminated simultaneously in runtime $O(|G_i|)$ by copying the corresponding elements of the function table. This results in runtime $O(|G_i|)$ per gate. Constant gates $G_{i'}$ with $d_{i'} = 0$ are propagated into their successors by recursively calling **eliminateConstInput(**G_s**,** k_s**,** $g_i(v_j)$**)** for all $G_s \in succ(G_{i'})$ having $G_{i'}$ as k_s-th input. If constant gate $G_{i'}$ is not an output gate it is eliminated afterwards.

In the example of Fig. 5, constant input in_3 is input into gate G_1 whose size is reduced by half when eliminating the second input ($k_1 = 2$). The resulting gate G'_1 has one non-constant input in_2 and hence no further optimization is performed. The other constant input in_7 is input into G_3 which is optimized into a constant gate G'_3 by eliminating the constant input. Hence, **eliminateConstInput()** is called recursively for successor G_5 and G'_3 is eliminated. Similarly to G_3, gate G_5 is reduced to a constant gate G'_5 and **eliminateConstInput()** is called for successor G_7 which eliminates its second input. As the output of G'_5 is also output of B it is not eliminated and remains as constant gate G_d.

After termination of Step 1 there might be gates G_i with one input left. The next two steps of Algorithm 1 try to remove these by incorporating their functionalities into their successors (Step 2) or predecessors (Step 3).

Step 2 - Eliminate non-output gates with one input. Step 2 of Algorithm 1 eliminates non-output gates with $d = 1$. The functionality of each one-input gate G_i which is not an output gate is incorporated into its successors $G_s \in succ(G_i)$ by the function **integrateInSucc(**G_i**)**. This function eliminates G_i by replacing it with a wire and incorporating the functionality of g_i into the function tables of all its successors $G_s \in succ(G_i)$: Let the output of G_i be the k-th input of G_s and d the degree of G_s. Then, the modified gate G'_s computes $g'_s(in_1, .., in_k, .., in_d) = g_s(in_1, .., g_i(in_k), .., in_d)$. Note that, independent of the functionality g_i, the resulting gate G'_s has the same size as G_s but additionally incorporates the functionality of g_i while not revealing any additional information on it. As in Step 1, for runtime $O(|G_i|)$ per gate the modifications of the function tables are not applied directly but first marked and then done simultaneously.

In the running example of Fig. 5, Step 2 eliminates G_1 by replacing it with a wire and modifying the function table of G_6 correspondingly. Analogously, gate G'_7 which only has one input from G_2 left after the optimizations performed in Step 1 is replaced by a wire. The function tables of its successors $G_9 \rightarrow G_b$ and $G_{10} \rightarrow G_c$ are modified correspondingly.

Step 3 - Eliminate output gates with one input. The third step of Algorithm 1 tries to eliminate output gates with $d = 1$. The functionality of each output gate G_i with one input is incorporated into its predecessor G_p. This is only possible if G_i is the only successor of G_p, i.e., $fanout(G_p) = 1$. In this case, function **integrateInPred(**G_i**,**G_p**)** is called which eliminates gate G_i by replacing it with a wire and incorporates its functionality into gate G_p with d inputs. The modified gate G'_p computes $g'_p(in_1, .., in_d) = g_i(g_p(in_1, .., in_d))$. As in Step 2, this optimization step is independent of the functionality g_i and the resulting gate G'_p has the same size as G_p but additionally incorporates the functionality of g_i while not revealing any additional information on it.

In the running example of Fig. 5, Step 3 eliminates G_8 by replacing it with a wire and modifying the function table of $G_6 \rightarrow G_a$ correspondingly. In contrast to this, gate G_c cannot be incorporated into its predecessor G_2 as G_c is not its only successor ($fanout(G_2) = 2$). The optimized block B_{opt} produced by Algorithm 1 is shown in Fig. 5(b). It has size $|B_{opt}| = 21$ which is less than 62% of the size of the original block $|B| = 34$.

Correctness, efficiency and security of Algorithm 1 are summarized in the following theorem. Its proof is given in the full version of this paper [15].

Theorem 1. *Algorithm 1 efficiently eliminates all $c > 0$ constant inputs that are not outputs of block B in runtime $O(|B|)$. The optimized block B_{opt} has smaller size and computes the same functionality as B. The topology of B_{opt} does not reveal anything about the values of the constant inputs.*

Acknowledgements. We would like to thank Vladimir Kolesnikov and anonymous reviewers of ACNS'09 for helpful comments on the paper.

References

1. Ahn, L.v., Hopper, N.J., Langford, J.: Covert two-party computation. In: ACM Symposium on Theory of Computing (STOC 2005), pp. 513–522. ACM Press, New York (2005)
2. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (2001)
3. Frikken, K.B., Atallah, M.J., Li, J.: Hidden access control policies with hidden credentials. In: ACM Workshop on Privacy in the Electronic Society (WPES 2004), p. 27. ACM Press, New York (2004)
4. Frikken, K.B., Atallah, M.J., Li, J.: Attribute-based access control with hidden policies and hidden credentials. IEEE Trans. Comput. 55(10), 1259–1270 (2006)
5. Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving credit checking. In: ACM conference on Electronic Commerce (EC 2005), pp. 147–154. ACM Press, New York (2005)
6. Frikken, K.B., Li, J., Atallah, M.J.: Trust negotiation with hidden credentials, hidden policies, and policy cycles. In: Network and Distributed System Security Symposium (NDSS 2006) (2006)
7. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)

8. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008), http://thomaschneider.de/FairplayPF
9. Laur, S., Lipmaa, H.: A new protocol for conditional disclosure of secrets and its applications. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 207–225. Springer, Heidelberg (2007)
10. Lindell, Y., Pinkas, B.: A proof of Yao's protocol for secure two-party computation. ECCC Report TR04-063, Electr. Coll. on Comp. Complexity (2004)
11. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
12. Lindell, Y., Pinkas, B., Smart, N.: Implementing two-party computation efficiently with security against malicious adversaries. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 2–20. Springer, Heidelberg (2008)
13. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay — a secure two-party computation system. In: USENIX (2004), http://www.cs.huji.ac.il/project/Fairplay/
14. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: ACM Symposium on Theory of Computing (STOC 1999), pp. 245–254. ACM Press, New York (1999)
15. Paus, A., Sadeghi, A.-R., Schneider, T.: Practical secure evaluation of semi-private functions. Cryptology ePrint Archive, Report 2009/124 (2009), http://eprint.iacr.org/
16. Sadeghi, A.-R., Schneider, T.: Generalized universal circuits for secure evaluation of private functions with application to data classification. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 336–353. Springer, Heidelberg (2008)
17. Valiant, L.G.: Universal circuits (preliminary report). In: Proc. 8th ACM Symp. on Theory of Computing (STOC 1976), pp. 196–203. ACM, New York (1976)
18. Yao, A.C.: How to generate and exchange secrets. In: Proc. 27th IEEE Symp. on Foundations of Comp. Science (FOCS 1986), Toronto, pp. 162–167. IEEE, Los Alamitos (1986)

Secure Hamming Distance Based Computation and Its Applications

Ayman Jarrous and Benny Pinkas*

University of Haifa

Abstract. This paper examines secure two-party computation of functions which depend only on the Hamming distance of the inputs of the two parties. We present efficient protocols for computing these functions. In particular, we present protocols which are secure in the sense of full simulatability against malicious adversaries.

We show different applications of this family of functions, including a protocol we call m-point-SPIR, which is an efficient variant of symmetric private information retrieval (SPIR). It can be used if the server's database contains N entries, at most $N/\log N$ of which have individual values, and the rest are set to some default value. This variant of PIR is unique since it can be based on the existence of OT alone.

1 Introduction

There are many known generic constructions of secure two-party and multi-party computation. It is preferable, of course, to use constructions which are secure against malicious adversaries, and where security is proved according to the full simulatability notion defined in [8]. In that case the composition theorem of [8] implies that the resulting protocol can be used as a building-block for more complex protocols, and security can be analyzed assuming that the building-block protocol is implemented by a trusted oracle [8,15]. There are recent efficient constructions of generic protocols which are secure according to this definition (by Lindell and Pinkas [22], and Jarecki and Shmatikov [20]), and there is even an implementation of the former protocol [23]. Our work investigates only the stand-alone setting, but there are also efficient generic constructions of secure two-party protocols in the UC model [19]. The downside of generic constructions is that they impose additional overheads, such as communicating and checking multiple copies of a circuit computing the functionality [22], or computing public key operations for every gate of the circuit [20]. It is therefore important to identify functionalities that are essential for many applications, and design efficient secure constructions of these specific functionalities. This paper performs this task for a functionality denoted as "Hamming distance based oblivious transfer", for which we also demonstrate different interesting applications.

* Supported by the the Israel Science Foundation (grant No. 860/06), the European Union under the FP7-STREP project CACE, and a European Research Council (ERC) Starting Grant.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 107–124, 2009.

The Hamming distance between two strings is defined as the number of characters in which they differ. We define "Hamming distance based oblivious transfer" (HDOT, pronounced "h-dot") as a protocol which allows two parties, a receiver $\mathcal{P}_1$ which has an input w, and a sender $\mathcal{P}_2$ which has an input w', to securely evaluate a function $f(\cdot,\cdot)$ whose output is determined only by the Hamming distance between w and w' (denoted $d_H(w,w')$). More precisely, the output is defined in the following way: Let $|w| = |w'| = \ell$, then $\mathcal{P}_2$ must provide $\ell+1$ additional inputs $Z_0,\ldots,Z_\ell$, and $\mathcal{P}_1$'s output is set to be Z_d where $d = d_H(w,w')$. With regards to this functionality, this paper contains the following results:

- HDOT protocols secure against semi-honest adversaries:
  - A protocol denoted *bin*HDOT for *binary* inputs $w, w' \in \{0,1\}^\ell$. This protocol operates by computing $O(\ell)$ homomorphic encryptions and only $\log \ell$ invocations of 1-out-of-2 oblivious transfer.
  - A general HDOT protocol, for $w, w' \in \Sigma^\ell$, where Σ can be arbitrary. This protocol uses *bin*HDOT as a building block.
- A *bin*HDOT protocol secure against malicious adversaries (in the stand-alone setting). The protocol uses two primitives that must also be secure against malicious adversaries: Committed Oblivious Transfer with Constant Difference (COTCD), and Oblivious Polynomial Evaluation (OPE). We give a construction for the first primitive, which is an example of a new class of OT protocols, *constrained OT*, which we define. The latter primitive is based on a construction of Hazay and Lindell [17].
- Applications of HDOT. These include several straightforward applications, such as computing the Hamming distance between strings, or transferring one of two words based on whether the two input strings are equal or not (a functionality we denote as *EQ*, for *equality based transfer*). Another application is a variant of symmetric PIR (SPIR) which we denote as m-point-SPIR, and which can be used when the server's database contains N items, of which at most $m = o(N/\log N)$ are unique and the other $N - m$ items have some default value. The receiver does not know whether it learns a unique or a default value. We show a protocol which is based on HDOT and can be reduced to oblivious transfer alone, which computes this functionality more efficiently than known PIR protocols. m-point-SPIR can be used for other applications, as described in Section 6.

2 Preliminaries

We use the standard definitions of secure two-party computation in the stand-alone setting (see Goldreich's book [15, Chapter 7]). Security of protocols is analyzed by comparing what an adversary can do in a real execution of the protocol to what it can do in an ideal scenario that is secure by definition. The ideal scenario involves an incorruptible trusted third party (TTP) which receives the inputs of the parties, computes the desired functionality, and returns to each party its respective output. A protocol is secure if any adversary which participates in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation. The exact definition appears in [15].

The hybrid model. Our protocols use other secure protocols, such as oblivious transfer, as subprotocols. It has been shown in [8] that if the subprotocols are secure according to the right definition (i.e., full simulatability in the case of the malicious adversary scenario), it suffices to analyze the security of the main protocol in a hybrid model. In this model the parties interact with each other and have access to a trusted party that computes for them the functionalities that are implemented by the subprotocols. The composition theorem states that it is not required to analyze the execution in the real model, but rather only compare the execution in the hybrid model to that in the ideal model.

2.1 Cryptographic Primitives and Tools

Homomorphic Encryption. A homomorphic encryption scheme allows to perform certain algebraic operations on an encrypted plaintext by applying an efficient operation to the corresponding ciphertext. In addition, we require in this paper that the encryption scheme be *semantically secure*. In particular, we use an additively homomorphic encryption schemes where the message space is a ring (or a field). There therefore exists an algorithm $+_{pk}$ whose input is the public key of the encryption scheme and two ciphertexts, and whose output is $E_{pk}(m_1 + m_2) = E_{pk}(m_1) +_{pk} E_{pk}(m_2)$. (Namely, given the public key alone this algorithm computes the encryption of the sum of the plaintexts of two ciphertexts.) The new ciphertext is an encryption which is done with fresh and independent randomness. There is also an efficient algorithm $\cdot_{pk}$, whose input consists of the public key of the encryption scheme, a ciphertext, and a constant c in the field, and whose output is $E_{pk}(c \cdot m) = c \cdot_{pk} E_{pk}(m)$.

An efficient implementation of an additive homomorphic encryption scheme with semantic security was given by Paillier [30,31]. In this cryptosystem the encryption of a plaintext from $[0, N-1]$, where N is an RSA modulus, requires two exponentiations modulo N^2. Decryption requires a single exponentiation. Security is based on the decisional composite residuosity (DCR) assumption.

Oblivious Transfer. The paper uses 1-out-of-N oblivious transfer (OT_1^N) as a basic building block. The OT_1^N protocol runs between two parties, a sender that has an input $(X_0, X_1, \ldots, X_{N-1})$, where $X_i \in \{0,1\}^m$, and a receiver that has an input $I \in \{0, 1, \ldots, N-1\}$. By the end of the protocol, the receiver learns X_I and nothing else and the sender does not learn any information about I. In [29] it was shown how to implement OT_1^N using $\log N$ invocations of OT_1^2. There are many efficient implementations of OT_1^2, starting with a protocol of Even, Goldreich and Lempel [10]. Most of these protocols are designed for the semi-honest scenario, or for a malicious scenario where the protocol provides only the privacy property and not full simulatability. We note that while our protocol for the semi-honest scenario can use any OT protocol, the protocol for the malicious adversary scenario must use an OT protocol which is secure in the sense of full simulatability against malicious adversaries. Such protocols were described, e.g., in [6,16,32,17]. (We specifically need a committed OT variant where we can also prove a relation between the inputs of the sender, and therefore we use a protocol which builds on the work of Jarecki and Shmatikov [20].) We also note that in the malicious case we use OT_1^2 and not OT_1^N.

2.2 Related Work

Generic secure computation. Generic protocols (e.g., of [35]) can be used to compute any function. They are typically based on representing the computed function as a binary or an algebraic circuit, and applying the protocol to this representation. The overhead of these protocols depends on the size of the circuit representation of the functions. There are many theoretical constructions of secure generic protocols. Notable examples of *implementations* of secure computation are the Fairplay system [24] for secure two-party computation, and the FairplayMP and SIMAP systems [1,3] for secure multi-party computation. The system described in [23] implements fully simulatable secure two-party computation according to the recent construction of [22].

Computing the Hamming distance. Protocols for computing the scalar product of vectors (which is equal to the Hamming distance if the alphabet is binary) were suggested in [34,14]. These protocols are based on the use of homomorphic encryption, and are only secure against semi-honest adversaries. (Our HDOT protocol for binary alphabets and semi-honest adversaries borrows its first step from these protocols.)

A protocol for secure efficient *approximate* computation of the Hamming distance, with a polylogarithmic communication overhead, was suggested in [18] (previous protocols for this task use $O(\sqrt{\ell})$ communication for ℓ-bit words [12,13]). We wanted to improve upon these protocols for three reasons: (1) These protocols introduce approximation errors. (2) The protocols are only secure against semi-honest adversaries. (3) In addition, these protocols have good asymptotic communication overhead, but use non-trivial components which seem difficult to implement with a performance that will be competitive for reasonable input sizes[1].

3 Hamming Distance Based Oblivious Transfer

A Hamming Distance based Oblivious Transfer protocol (abbrev. HDOT) is run between two parties, a receiver ($\mathcal{P}_1$) and a sender ($\mathcal{P}_2$). It is defined as follows:

- *Input:* $\mathcal{P}_1$'s input is a word $w \in \Sigma^\ell$. $\mathcal{P}_2$'s input contains a word $w' \in \Sigma^\ell$, and $\ell + 1$ values $Z_0, \ldots, Z_\ell$.
- *Output:* $\mathcal{P}_1$'s output is Z_d, where $d = d_H(w, w')$ is the Hamming distance between w and w' (note that $\mathcal{P}_1$ does not learn the Hamming distance itself). $\mathcal{P}_2$ has no output.

[1] For example, the protocol in [18] applies the Naor-Nissim [27] protocol to a circuit which computes vector operations over the Real numbers and samples from a Bernoulli distribution; in addition it uses symmetric PIR protocols.

The paper describes a special protocol, *bin*HDOT, for the case that the input words are binary (i.e., $\Sigma = \{0, 1\}$), and a general protocol which works for alphabets Σ of arbitrary size.

3.1 Straightforward Applications

An HDOT protocol can be immediately used for computing any function which depends on the Hamming distance. Following are some interesting examples of such functions:

- The *Hamming distance* itself can be computed by setting $Z_i = i$ for every $0 \le i \le \ell$.
- The *parity* of the exclusive-or of the two inputs is computed by setting Z_i to be equal to the least significant bit of i, for $0 \le i \le \ell$.
- *EQ – Equality based transfer, or* $EQ_{\mathcal{V}_0,\mathcal{V}_1}(w, w')$*:* This functionality outputs $\mathcal{V}_0$ if $w = w'$, and $\mathcal{V}_1$ otherwise. The functionality is computed by setting $Z_0 = \mathcal{V}_0$ and $Z_i = \mathcal{V}_1$ for $1 \le i \le \ell$, and executing an HDOT protocol. $\mathcal{P}_1$ does not know which of the two cases happens (namely, whether $w = w'$). This is crucial for the applications that are described below.

  Recall that it is easy to design a protocol in which $\mathcal{P}_1$ learns a specific value $\mathcal{V}_0$ if the two inputs are equal, and a *random* value otherwise. (See [11], or consider a protocol where $\mathcal{P}_1$ sends a homomorphic encryption $E(w)$, and receives back $E(r \cdot (w - w') + \mathcal{V}_0)$, where r is a random value.) Our protocol is unique in defining a specific value to be learned if the two inputs are different, and in hiding whether the inputs are equal or not.[2]
- *Threshold HDOT protocol:* The equality based transfer protocol can be generalized to tolerate some errors and have the output be $\mathcal{V}_0$ if the Hamming distance is smaller than a threshold τ, and be $\mathcal{V}_1$ otherwise. In other words, it implements the following functionality:

$$\mathrm{HDOT}^{\tau}_{\mathcal{V}_0|\mathcal{V}_1}(w, w') = \begin{cases} \mathcal{V}_0, \, d_H(w, w') < \tau \\ \mathcal{V}_1, \, d_H(w, w') \ge \tau \end{cases}$$

  This functionality is implemented by setting $Z_0 = \cdots = Z_{\tau-1} = \mathcal{V}_0$, and $Z_\tau = \cdots = Z_\ell = \mathcal{V}_1$.

The protocol for equality based transfer is the major building blocks of the m-point-SPIR SPIR application described in Section 6.

4 Protocols Secure against Semi-honest Adversaries

We first describe protocols which are secure against semi-honest behavior of the potential adversaries. These protocols are relatively simple, yet they are unique in invoking oblivious transfer a number of times which is only logarithmic in the input length. The malicious adversary scenario is covered in Section 5.

[2] In [2] it was shown how to implement a protocol which transfers one of two strings if $w > w'$, and transfers the other string if $w < w'$ (if $w = w'$ the output is random). It is possible to compute the EQ functionality by combining that protocol with a protocol which outputs a specific value if $w = w'$ and a random value otherwise.

4.1 A Protocol for Binary Alphabets (*bin*HDOT)

Consider first the case where the alphabet is binary ($\Sigma = \{0,1\}$). The *bin*HDOT functionality can be securely implemented by applying Yao's protocol to a circuit computing it. That solution would require running ℓ invocations of OT_1^2. We describe here a protocol which accomplishes this task using only $\log(\ell+1)$ OT_1^2s (see below a comparison of the performance of these two protocols).

The protocol works in the following way: In the first step the parties use homomorphic encryption to count the number of bits in which the two words differ. The result is in the range $[0,\ell]$. Next, the two parties use $\mathrm{OT}_1^{\ell+1}$ (implemented using $\log(\ell+1)$ OT_1^2s) to map the result to the appropriate output value. The protocol is described in detail in Figure 1.

Correctness. The value d_H is equal to the Hamming distance. In Step 4, $\mathcal{P}_1$ computes (in $\mathcal{F}$) the value $d_H + r$, which can have one of $\ell+1$ values (namely $r, r+1, \ldots, r+\ell$). It holds with probability $1 - \ell/|\mathcal{F}|$ that $r < |\mathcal{F}| - \ell$. (And

$bin\mathrm{HDOT}_{\langle Z_0,\ldots,Z_\ell\rangle}(w,w')$ PROTOCOL

INPUT: $\mathcal{P}_1$'s input is a word $w = (w_0,\ldots,w_{\ell-1})$, $\mathcal{P}_2$'s input is $w' = (w'_0,\ldots,w'_{\ell-1})$, where $w_i, w'_i \in \{0,1\}$. $\mathcal{P}_2$ has additional inputs $(Z_0,\ldots,Z_\ell)$.

OUTPUT: $\mathcal{P}_1$ receives Z_i such that $d_H(w,w') = i$. $\mathcal{P}_2$ learns nothing.

The protocol uses $E_{pk}(\cdot)$, a homomorphic encryption function. The plaintexts are in a ring or a field $\mathcal{F}$. (We emphasize that ℓ and $|\Sigma|$ are negligible compared to $|\mathcal{F}|$. A typical size could be $|\mathcal{F}| = 2^{1024}$.) *pk* is a public key that both parties know, but only $\mathcal{P}_1$ knows the corresponding private key and can decrypt messages.

1. $\mathcal{P}_1$ sends the homomorphic encryption of each bit of the binary representation of $w = \{w_0,\ldots,w_{\ell-1}\}$, where $w_i \in \{0,1\}$.
2. $\mathcal{P}_2$ receives the encrypted representation $\{E_{pk}(w_0),\ldots,E_{pk}(w_{\ell-1})\}$. For each bit location j it calculates $E_{pk}(\vartheta_j)$, where $\vartheta_j \in \{0,1\}$ and is equal to 1 if, and only if, $w_j \neq w'_j$. The calculation is done in the following way:
$$E_{pk}(\vartheta_j) = E_{pk}(w_j) \cdot_{pk} (1 - w'_j) +_{pk} (1 -_{pk} E_{pk}(w_j)) \cdot_{pk} w'_j$$
3. Using the homomorphic properties, $\mathcal{P}_2$ sums the results of the previous step and computes $E_{pk}(d_H) = \sum_0^{\ell-1} E_{pk}(\vartheta_i)$. The value d_H is in the range $\{0,1,\ldots,\ell\}$ and is equal to the Hamming distance between the two input words. In addition, $\mathcal{P}_2$ chooses a random value $r \in \mathcal{F}$, computes the value $E_{pk}(d_H + r)$, and sends it to $\mathcal{P}_1$. (In other words, it shifts the result by a random value r. Note that with overwhelming probability, $1 - \ell/|\mathcal{F}|$, this addition operation does not involve a modular reduction.)
4. $\mathcal{P}_1$ receives $E_{pk}(d_H + r)$ and decrypts the result.
5. Next, the parties map the result to the appropriate Z_i value, by invoking a $\mathrm{OT}_1^{\ell+1}$ protocol where $\mathcal{P}_1$ is the receiver and $\mathcal{P}_2$ is the sender:
   - The input of $\mathcal{P}_1$ is $(d_H + r)\ mod\ (\ell+1)$.
   - $\mathcal{P}_2$ has inputs $X_0,\ldots,X_\ell$, where $X_i = Z_{(i-r)\ mod\ (\ell+1)}$ (namely, Z_i is mapped to input $(i+r)\ mod\ (\ell+1)$ of the OT).

   $\mathcal{P}_1$'s output in the OT is its output in the *bin*HDOT protocol.

Fig. 1. The *bin*HDOT protocol

since $|\mathcal{F}|$ is typically very large compared to ℓ, e.g. $|\mathcal{F}| \approx 2^{1024}$ and $\ell < 1000$, we do not consider here the negligible probability that this event does not happen.) Therefore, the computation of $d_H + r$ in $\mathcal{F}$ does not involve a modular reduction and has the same result as adding them over the integers. Reducing the result modulo $\ell + 1$ (in Step 5) is therefore equal to $(r + d_H) \bmod (\ell + 1)$. $\mathcal{P}_1$ uses this result as its input to the 1-out-of-$(\ell + 1)$ OT protocol of Step 5. $\mathcal{P}_2$, on the other hand, sets the sender's inputs in the OT such that each Z_i value is the sender's input indexed by $(r + i) \bmod (\ell + 1)$. As a result, the output of $\mathcal{P}_1$ in the OT protocol is Z_{d_H}, as required.

Note that if the parties are only interested in computing the value of the Hamming distance then the protocol can be greatly simplified: $\mathcal{P}_2$ should send to $\mathcal{P}_1$ in Step 3 the encryption $E_{pk}(d_H)$. There is no need to run Steps 4 and 5.

Improving the initial step using non-interactive preprocessing. An additional improvement can be achieved in the first step of the protocol, where $\mathcal{P}_1$ sends an encrypted binary representation of the word. This representation can be precomputed using *non-interactive* preprocessing: $\mathcal{P}_1$ can prepare in advance ℓ encrypted zeros and ℓ encrypted ones, instead of encrypting the input bits online. This preprocessing enables $\mathcal{P}_1$ to send the binary representation directly without spending time online encrypting 0 and 1 values.

Overhead. We compare the overhead of the *bin*HDOT protocol to that of applying Yao's protocol to a circuit computing the same functionality. We note that the runtime of an OT protocol is slower than that of a homomorphic encryption or decryption, and that the runtime of these latter operations is *much* slower than that of a homomorphic addition or a homomorphic multiplication by a constant (which in turn is *much* slower than symmetric encryption or decryption). This relation between run times can be summarized as follows (where $>$ denotes "slower", and $\gg$ denotes slower by an order of magnitude):

$$\text{OT} > \text{homomorphic enc.} \gg \text{homomorphic addition} \gg \text{symmetric enc.}$$

Without using any preprocessing, the *bin*HDOT protocol requires $\mathcal{P}_1$ to compute ℓ encryptions and a single decryption, while $\mathcal{P}_2$ computes $\ell+1$ homomorphic additions, and the two parties run $\log(\ell+1)$ OT_1^2s and $2(\ell+1)$ symmetric encryptions (in order to implement $\text{OT}_1^{\ell+1}$). In Yao's protocol, the parties compute a circuit with ℓ input bits and a total of $O(\ell)$ gates. This requires ℓ executions of an OT_1^2 protocol and $O(\ell)$ symmetric encryptions and decryptions. Both protocols require $O(\ell)$ communication.

The improvement achieved by the *bin*HDOT protocol is noticeable since it reduces the number of OTs, which are the most time consuming operation, from ℓ to $\log(\ell+1)$. In addition, the *bin*HDOT protocol can benefit from the use of *non-interactive* preprocessing to precompute all homomorphic encryption operations even before the parties know of each other. In that case the ℓ encryptions done by $\mathcal{P}_1$ are computed offline, and its online computation is composed of a single decryption and $\log(\ell + 1)$ OTs. (Yao's protocol cannot precompute the oblivious transfers without using interaction. We note that if interactive preprocessing is possible, then the OTs themselves can be precomputed in both protocols, and this reduces the overhead of both protocols.)

Security. (sketch) We analyze security assuming that the parties are semi-honest. The proof is simple, and therefore we only give a sketch of the proof: We assume that the OT protocol is secure, and therefore we can prove security in a hybrid model where the OT protocol is implemented by an oracle. In the protocol, $\mathcal{P}_2$ receives homomorphic encryptions of a binary representation of a word, and then it plays the role of the sender in an OT protocol in which it receives no output. Therefore, if $\mathcal{P}_2$ learns anything this information must have leaked from the encryptions it received. In other words, it is easy to write a reduction showing that any algorithm that $\mathcal{P}_2$ might use to learn information can be used to break the security of the semantic security of the encryption. $\mathcal{P}_1$ receives from $\mathcal{P}_2$ a *random* value $(d_H + r)$. It then participates in the OT protocol, which we assume to be implemented by an oracle. $\mathcal{P}_2$ therefore learns nothing but the output of the OT, which is its designated output of the protocol.

4.2 A Protocol for Arbitrary Alphabets (HDOT)

We now describe an HDOT protocol which works over arbitrary alphabets Σ. The protocol is based on applying the *bin*HDOT protocol to every character of the words. More specifically, the parties have inputs $w, w' \in \Sigma^\ell$, respectively. The protocol begins with the parties representing each of the letters of Σ as a binary word of length $\lceil \log |\Sigma| \rceil$, and then running (for each letter location) the equality based transfer (EQ) protocol, which was defined above and is an application of *bin*HDOT. In each execution of the EQ protocol $\mathcal{P}_1$ learns a value α_i if $w_i = w'_i$, or the value $\alpha_i + 1$ otherwise, where α_i is chosen at random by $\mathcal{P}_2$. Then, $\mathcal{P}_1$ sums the values that it has received modulo $\ell + 1$. The result is equal, modulo $\ell + 1$, to $\sum \alpha_i$ plus the Hamming distance of the original words. The parties then run an $\mathrm{OT}_1^{\ell+1}$ protocol to map the result to the desired output. The protocol is detailed in Figure 2.

Correctness. For every $0 \leq i \leq \ell - 1$, $\mathcal{P}_1$ and $\mathcal{P}_2$ learn in Step 1 values β_i, α_i, respectively, such that $\beta_i = \alpha_i$ if the letters w_i and w'_i are equal, and $\beta_i = \alpha_i + 1$ if the letters are different. Let $S_\alpha = \sum_{i=0}^{\ell-1} \alpha_i$, where here the addition is done in $\mathcal{F}$. Define S_β similarly. Let d be the Hamming distance between the two input words. Then it holds with probability $1 - \ell/|\mathcal{F}|$ that $S_\beta = S_\alpha + d$, where the addition here is done over the integers. Therefore, the values $\sigma_\alpha = S_\alpha \bmod (\ell{+}1)$ and $\sigma_\beta = S_\alpha \bmod (\ell + 1)$ computed in Step 2 satisfy that $\sigma_\beta - \sigma_\alpha \bmod (\ell + 1)$ is equal to the Hamming distance d (which is in the range $[0, \ell]$).

Consider now the OT in Step 3. Assume first that $\sigma_\alpha = 0$. In this case $\mathcal{P}_1$'s input to the OT, σ_β, is equal to the Hamming distance, and the inputs of $\mathcal{P}_2$ to the OT are the values $Z_0, \ldots, Z_\ell$ (in that order). The OT protocol therefore computes the desired output in this case. Now, if $\sigma_\alpha > 0$ then $\mathcal{P}_1$'s input to the OT protocol is cyclically shifted (modulo $\ell + 1$) by σ_α, while the order of $\mathcal{P}_2$'s inputs to the OT is also cyclically shifted (modulo $\ell + 1$) by the same value σ_α. The OT protocol therefore computes the correct result.

HDOT$_{\langle Z_0,\ldots,Z_{\ell+1}\rangle}(w,w')$ PROTOCOL

INPUT: $\mathcal{P}_1$ has an input $w = \langle w_0, w_1, \ldots, w_{\ell-1}\rangle \in \Sigma^\ell$. $\mathcal{P}_2$ has an input $w' = \langle w'_0, w'_1, \ldots, w'_{\ell-1}\rangle \in \Sigma^\ell$, and additional input values $Z_0, \ldots, Z_\ell$. We denote by $\bar{w_j}$ the binary representation of w_j, which is $\lceil \log(|\Sigma|)\rceil$ bits long.
OUTPUT: $\mathcal{P}_1$ learns Z_i such that $d_H(w,w') = i$, $\mathcal{P}_2$ learns nothing.

1. For every $i \in [0, \ell-1]$, $\mathcal{P}_2$ chooses at random a value $\alpha_i \in_R \mathcal{F}$. Both parties then run the protocol $\text{EQ}_{\alpha_i,\alpha_i+1}(\bar{w_i}, \bar{w_i}')$. ($\bar{w_i}, \bar{w_i}'$ denote the binary representations of the letters w_i and w'_i, respectively. The output of this protocol is α_i if $w_i = w'_i$, and $\alpha_i + 1$ otherwise.)
   At the end of the process, $\mathcal{P}_1$ obtains the values $\{\beta_0, \ldots, \beta_{\ell-1}\}$, where
   $$\beta_i = \begin{cases} \alpha_i, & w_i = w'_i \\ \alpha_i + 1, & w_i \neq w'_i \end{cases}$$
2. $\mathcal{P}_1$ sums, modulo $\ell + 1$, the β_i values it received. Namely, it computes $\sigma_\beta = (\sum_0^{\ell-1} \beta_i) \bmod (\ell+1)$. $\mathcal{P}_2$ sums its α values and computes $\sigma_\alpha = (\sum_0^{\ell-1} \alpha_i) \bmod (\ell+1)$.
3. Both parties run an $\text{OT}_1^{\ell+1}$ protocol with the following inputs:
   - $\mathcal{P}_1$ is the receiver and its input is σ_β.
   - $\mathcal{P}_2$ is the sender and its input is $\{X_0, \ldots, \ldots, X_\ell\}$, where $X_i = Z_{(i-\sigma_\alpha)\ mod\ (\ell+1)}$.

   The value that $\mathcal{P}_1$ receives in the OT is defined as its output in the protocol.

Fig. 2. The HDOT protocol for general alphabets

Overhead. The overhead is that of applying the *bin*HDOT protocol ℓ times over $\log|\Sigma|$ long binary strings, and then running $\log(\ell+1)$ invocations of OT_1^2. The parties run $\ell \log\log|\Sigma| + \log(\ell+1)$ OT_1^2s. (A direct implementation of this functionality using Yao's protocol would have required invoking $O(\ell \log|\Sigma|)$ OTs.)

Security. (sketch) Analyzing security in the hybrid model, we assume that the *bin*HDOT and OT protocols are executed by a trusted oracle. Then $\mathcal{P}_2$, being the sender in these protocols, cannot learn any information about the input of $\mathcal{P}_1$. $\mathcal{P}_1$ receives the β_i values in the first step, but it cannot distinguish whether $\beta_i = \alpha_i$ or $\beta_i = \alpha_i + 1$, since each α_i value was chosen randomly by $\mathcal{P}_2$. In the last step, $\mathcal{P}_1$ receives the result of mapping the sum of the β values to the appropriate Z_i value, which is also the result it would have received from the trusted party.

4.3 Weighted Hamming Distance Based OT

The *weighted* Hamming distance between two ℓ-letter strings w, w' is defined in the following way: The function depends on a set of integer weights $\omega_0, \ldots, \omega_{\ell-1}$. We define δ_i, for $0 \leq i \leq \ell - 1$, to be 0 if $w_i = w'_i$, and 1 otherwise. The weighted Hamming distance is $\sum_{i=0}^{\ell-1} \delta_i \omega_i$ (earlier we handled the case where

$\forall i \quad \omega_i = 1$). This function enables to assign to any letter location a specific weight corresponding to its importance.

It is possible to slightly change the HDOT protocols to support the computation of a weighted Hamming distance based OT. In the binary alphabet case, the revised *bin*HDOT protocol computes in Step 2 the values $E_{pk}(\vartheta_j \omega_j)$ by multiplying $E_{pk}(\vartheta_j)$ by ω_i. The value d_H is defined to be the sum of these values. Let $\Omega = \sum_{i=0}^{\ell-1} \omega_i$. The value of d_H is in the range $[0, \Omega]$. Therefore $\mathcal{P}_2$ has inputs $Z_0, \ldots, Z_\Omega$, and the last step of the protocol computes a 1-out-of-($\Omega + 1$) OT. In the case of an arbitrary alphabet, each β_i value is set to $\alpha_i + \omega_i$ if the two letters are different, and to α_i is they are equal. Again, the last step computes a 1-out-of-($\Omega + 1$) OT.

5 A *bin*HDOT Protocol for Malicious Adversaries

We design a new *bin*HDOT protocol to handle the presence of malicious adversaries. In this protocol the parties use a new variant of OT_1^2 to learn whether corresponding bits of the two words are equal, and then use an Oblivious Polynomial Evaluation (OPE) protocol [28,17] to map the result to an output value. (This is different than the semi-honest case, where homomorphic encryption was used to compare bits, and OT_1^N was used to compute the final result.) The new protocol uses OT and OPE protocols which are efficient and yet are secure in the sense of full simulatability against malicious adversaries. Security can therefore be analyzed in the hybrid model. In more detail, the protocol uses the following tools:

Committed 1-out-of-2 Oblivious Transfer with Constant Difference (or COTCD_1^2), secure against malicious adversaries. A committed OT protocol in an OT protocol where the parties commit to their inputs: the sender commits to its inputs m_0, m_1 and the receiver commits to its input $\sigma \in \{0, 1\}$. During the protocol each party can verify that the other party's input is equal to the corresponding committed value. We define a committed OT with constant difference (COTCD, pronounced "cot-cd") to be a committed OT with an additional auxiliary input composed of a value Δ known to the sender, and a commitment to Δ which is known to the receiver. The protocol lets the receiver verify that the difference of the two inputs of the sender is $\pm\Delta$. In other words, it either holds that $m_1 - m_0 = \Delta$ or that $m_0 - m_1 = \Delta$.

We use a COTCD primitive which is based on the Jarecki and Shmatikov (JS) committed OT protocol [20], which is in turn based on the Camenisch-Shoup (CS) encryption scheme [7]. The details of the COTCD protocol are described in the full version of our paper.[3] We use that protocol since it can be used to transfer

[3] The COTCD protocol is identical to the Jarecki and Shmatikov (JS) protocol [20], with an addition of a preliminary step and a verification step. In the preliminary step, both parties receive their auxiliary inputs: the sender receives a value Δ, which is the difference that must hold between its input values, and the receiver receives the committed value of Δ. In the verification step the sender proves to the receiver in zero-knowledge that the committed values, m_0, m_1, have a difference $\pm\Delta$. It is important to note that the receiver knows only $\mathsf{Com}(\Delta)$ and does not learn Δ.

strings, and since it is easy to add to it an efficient zero-knowledge proof that the messages of the sender have the required difference (it seems much harder to add a proof of this type to other OT protocols which are secure against malicious adversaries, such as the protocols of [17,32]). The JS protocol is UC-secure in the common reference string model and therefore all invocations of that protocol can be run in parallel (as a result, the HDOT protocol we construct can execute in parallel all ℓ invocations of the COTCD protocol). The protocol is proved to be secure under the decisional composite residuosity (DCR) assumption (i.e., the assumption on which the Paillier homomorphic encryption system is based).

Commitment scheme. The Camenisch-Shoup (CS) encryption scheme [7] is used in our protocol as a commitment scheme, as is suggested in [20].

An Oblivious Polynomial Evaluation (OPE) protocol secure against malicious adversaries. An OPE protocol [28] is a protocol where the sender's input is a polynomial $P()$ of a certain degree, and the receiver's input is a value x. The receiver's output is $P(x)$ while the sender learns nothing. We use the OPE construction of Hazay and Lindel [17], which is secure (in the sense of full simulatability) against malicious adversaries, and uses very few exponentiations.

The underlying fields. The output of the COTCD protocol is used as an input of the OPE protocol. The COTCD protocol runs in a group $\mathcal{F} = \mathbb{Z}^*_{n^2}$, where $\mathbb{Z}^*_{n^2}$ is defined by a safe RSA modulus $n = pq$. The encryption scheme of Camenisch and Shoup, which is used in the protocol as a commitment scheme, works in the same group. The OPE protocol of [17] runs in $\mathbb{Z}_N$, with N being an RSA modulus. Our protocol must enable the parties to use the result of the COTCD protocol as an input to the OPE protocol. It must therefore use a group $\mathbb{Z}^*_{n^2}$ and a field $\mathbb{Z}_N$, which satisfy that $|\mathbb{Z}^*_{n^2}| < |\mathbb{Z}_N|$, and therefore we will require that $n^2 < N$. We define a simple mapping $f : \mathbb{Z}^*_{n^2} \to \mathbb{Z}_N$, where the only requirement is that no two elements of $\mathbb{Z}^*_{n^2}$ are mapped by f to the same value in $\mathbb{Z}_N$. The protocol then performs the initial computations in $\mathbb{Z}^*_{n^2}$ and then uses f to map the result to $\mathbb{Z}_N$.

The protocol itself is described in Figure 3. In the protocol, for every bit location i $\mathcal{P}_1$ receives a value t^0_i if the corresponding bits are equal, and the value $t^0_i + \Delta$ otherwise. The value Δ, and also all t^0_i values, are randomly chosen by $\mathcal{P}_2$. (In the semi-honest case $\mathcal{P}_1$ learned one of two values whose difference was 1. Here the difference is a random number Δ in order to prevent attacks by a malicious $\mathcal{P}_1$.) $\mathcal{P}_1$ then sums the values it received, and obtains the result $\sum_{i=1}^{\ell} t^0_i + d \cdot \Delta$, where d is the Hamming distance. We use the notation $\sigma_r = \sum_{i=1}^{\ell} t^0_i$. $\mathcal{P}_2$ then prepares an OPE where $\forall j \in [0, \ell]$, $P(\sigma_r + j \cdot \Delta) = Z_j$. The parties execute an OPE and $\mathcal{P}_1$ computes $P(\sigma_r + d\Delta)$ and learns the desired result.

The protocol uses an OPE instead of $\mathrm{OT}^{\ell+1}_1$ since the values are mapped to locations in a large range, rather than to indices in the range $[0, \ell]$, in order to prevent a malicious $\mathcal{P}_1$ from learning any Z_i value which does not correspond to the actual Hamming distance. If $\mathcal{P}_1$ evaluates the polynomial at any point other than intended, it is likely to receive a random answer since it does not know Δ and is therefore unlikely to choose any point corresponding to a Z_i value. As for

Input: $\mathcal{P}_1$'s input is a word $w = (w_0, \ldots, w_{\ell-1})$, $\mathcal{P}_2$'s input is $w' = (w'_0, \ldots, w'_{\ell-1})$, where $w_i, w'_i \in \{0,1\}$. $\mathcal{P}_2$ has additional inputs $(Z_0, \ldots, Z_\ell)$.
Output: $\mathcal{P}_1$ receives Z_i such that $d_H(w, w') = i$ (i.e. the Hamming distance of w and w' is i). $\mathcal{P}_2$ learns nothing.

1. $\mathcal{P}_2$ chooses at random $\Delta \in_R \mathbb{Z}^*_{n^2}$ and sends to $\mathcal{P}_1$ a commitment to Δ. In addition it proves to $\mathcal{P}_1$, using a zero-knowledge proof of knowledge, the knowledge of Δ.
2. For each pair of bits (w_i, w'_i), both parties use COTCD to check whether the bits are equal:
   - $\mathcal{P}_2$ chooses a random value $t_i^0 \in_R \mathcal{F}$, and defines $t_i^1 = t_i^0 + \Delta$.
   - Both parties run a COTCD protocol:
     (a) The auxiliary inputs to the protocol are Δ, known to $\mathcal{P}_2$, and a commitment to Δ, known to $\mathcal{P}_1$.
     (b) $\mathcal{P}_1$ is the receiver and its input is w_i.
     (c) $\mathcal{P}_2$ is the sender. If $w'_i = 0$ then it sets $(x_i^0, x_i^1) = (t_i^0, t_i^1)$; Otherwise, $(x_i^0, x_i^1) = (t_i^1, t_i^0)$.

   In each execution of the protocol, if both bits are equal then $\mathcal{P}_1$ learns t_i^0, otherwise, $\mathcal{P}_1$ learns t_i^1. (If $|x_i^1 - x_i^0| \neq \Delta$ then $\mathcal{P}_1$ aborts.)

   By the end of this step, $\mathcal{P}_1$ learns $t_0^{b_0}, \ldots, t_{\ell-1}^{b_{\ell-1}}$, where $b_i = w_i \oplus w'_i$, while $\mathcal{P}_2$ does not learn any information.
3. $\mathcal{P}_1$ computes $\sigma_t = \sum t_i^{b_i}$ and $\mathcal{P}_2$ computes $\sigma_r = \sum t_i^0$. These summations are done in $\mathbb{Z}^*_{n^2}$.
4. $\mathcal{P}_2$ constructs a polynomial $P(x) = \sum_0^\ell a_i x^i$ in $\mathbb{Z}_N$, such that $P(f(\sigma_r + i \cdot \Delta)) = Z_i, \forall i \in \{0, 1, \ldots, \ell\}$ (where f is the simple mapping from $\mathbb{Z}^*_{n^2}$ to $\mathbb{Z}_N$), and $P(0)$ is random. (This construction succeeds if $0 \notin \{\sigma_r, \ldots, \sigma_r + \ell\Delta\}$, which happens with probability $1 - (\ell+1)/|\mathbb{Z}_N|$.) The degree of P is $\ell + 1$.
5. $\mathcal{P}_1$ and $\mathcal{P}_2$ run an OPE protocol to evaluate $P(f(\sigma_t))$, such that $\mathcal{P}_1$ learns the result while $\mathcal{P}_2$ does not learn any information.

Fig. 3. The *bin*HDOT protocol for the malicious case

a malicious $\mathcal{P}_2$, its inputs w' and $Z_0, \ldots, Z_\ell$ can be extracted from its interaction with the OT and OPE protocols, and are used for a simulation based proof.

Theorem 1. *The protocol computes the binHDOT functionality.*

Proof. Let us follow the steps of the protocol. In each execution of the COTCD protocol, $\mathcal{P}_1$ learns t_i^0 if both bits are equal, otherwise, it learns $t_i^1 = t_i^0 + \Delta$. In other words, it learns $t_i^{b_i}$, where $b_i = w_i \oplus w'_i$. Then, in Step 3, $\mathcal{P}_1$ computes $\sigma_t = t_0^{b_0} + \cdots + t_{\ell-1}^{b_{\ell-1}}$, and $\mathcal{P}_2$ computes $\sigma_r = t_0^0 + \cdots t_{\ell-1}^0$. Therefore it holds that $\sigma_t - \sigma_r = \Delta \cdot d_H(w, w')$. In Step 4, $\mathcal{P}_2$ constructs a polynomial $P(x)$ such that: $P(f(\sigma_r)) = Z_0$; $P(f(\sigma_r + \Delta)) = Z_1$;$\ldots$; $P(f(\sigma_r + \ell \cdot \Delta)) = Z_\ell$. In the last step of the protocol, the parties use an OPE protocol to compute $P(f(\sigma_t)) = Z_{d_H(w,w')}$.

Theorem 2. *The protocol securely computes binHDOT in the presence of malicious adversaries.*

Proof. (Sketch) The security of the protocol is proved in the hybrid model, assuming that the COTCD and OPE primitives, as well as the zero-knowledge proof of knowledge of Δ used in the protocol, are performed by a trusted oracle (or trusted party). This assumption is justified since, as we detailed above, there are constructions of these primitives which have fully simulatable security against malicious adversaries (where the security is based on the Decisional Composite Residuosity (DCR) assumption).

We compare the execution of the protocol between $\mathcal{P}_1$ and $\mathcal{P}_2$ to an execution with a *trusted third party (TTP)*, where the TTP receives the inputs of both parties and computes the following functionality: If the input of $\mathcal{P}_1$ is w and the input of $\mathcal{P}_2$ is $\langle w', Z_0, \ldots, Z_\ell \rangle$, then the output of $\mathcal{P}_1$ is $Z_{d_H(w,w')}$. Otherwise if the input of $\mathcal{P}_1$ is a special symbol ρ then the output of $\mathcal{P}_1$ is a random value; otherwise if the input of either party is a special symbol $\perp$ then the protocol terminates prematurely.

We first prove security in the case that $\mathcal{P}_1$ is corrupt and then in the case that $\mathcal{P}_2$ is corrupt.

$\mathcal{P}_1$ is corrupt. The full proof appears in the full version of the paper. The idea behind the proof is that $\mathcal{P}_1$'s choices in the COTCD protocols define its input w. Then, $\mathcal{P}_1$ is supposed to add the values it received in the COTCD invocations and use the result as its input to the OPE. If it uses a different input to the OPE protocol, then, since it does not know Δ, it happens with overwhelming probability that $\mathcal{P}_1$ queries a value of the polynomial at a point which was not defined by $Z_0, \ldots, Z_\ell$ and receives a random answer.

$\mathcal{P}_2$ is corrupt. The full proof appears in the full version of the paper. The proof is based on the following ideas: (1) the simulator extracts the value of Δ from the zero-knowledge proof of knowledge; (2) the simulator then learns the inputs that $\mathcal{P}_2$ uses in the COTCD invocations, and based on these values the simulator computes w' and σ_r; (3) it also learns the coefficients of the polynomial $P()$ which is $\mathcal{P}_2$'s input to the OPE, and can therefore compute $Z_0 = P(\sigma_r), \ldots, Z_\ell = P(\sigma_r + \ell\Delta)$; (4) finally, the simulator sends $\langle w', Z_0, \ldots, Z_\ell \rangle$ to the TTP.

Efficiency. The overhead of the protocol is composed of running ℓ invocations of the COTCD protocol (which can be run in parallel, since the protocol is UC-secure), and a single invocation of the OPE protocol of [17]. Both of these protocol can be run in a constant number of rounds.

5.1 Securing the Applications against Malicious Adversaries

The protocol described above is secure against malicious behavior of either party. However, it does not enforce any structure of the inputs $Z_0, \ldots, Z_\ell$ of $\mathcal{P}_2$ and therefore a corrupt $\mathcal{P}_2$ can set these inputs to arbitrary values. This "feature" does not affect plain usage of the protocol, but it means that security against malicious adversaries cannot be guaranteed if the protocol is used for computing any functionality that requires specific relations between the Z_i values. Unfortunately, this is relevant to the relations required in the applications detailed in Section 3.1. For example, the EQ application, i.e., equality based transfer, requires that $Z_1 = Z_2 = \cdots = Z_\ell$. As a result, the protocol cannot be used "as is" as a building block for protocols (secure against malicious adversaries) for the HDOT functionality for arbitrary alphabets, or for the EQ functionality.

In order to adapt the protocol for these tasks, it is required to add zero-knowledge proofs which assure $\mathcal{P}_1$ that the Z_i inputs follow the desired structure. This is of course possible in principle, but in this work we have not examined how to optimize the efficiently of such proofs. We will only describe here the steps which are required in order to design and implement an EQ protocol secure against malicious adversaries (protocols for the other applications can be designed in a similar way): (1) The protocol needs an additional step where $\mathcal{P}_1$ obtains a commitment $\mathsf{Com}(\sigma_r)$ to $\sigma_r = \sum t_i^0$. This commitment can be computed given the commitments that $\mathcal{P}_2$ generates in the committed OT protocols; the correctness of the committed value can be proved using $\mathcal{P}_2$'s proofs about the Δ differences of its input pairs. (Namely, $\mathcal{P}_2$ must prove that there exist bits $b_0, \ldots, b_{\ell-1}$ such that $\sum x_i^{b_i} = \sigma_r$, and that $\forall i \; x_i^1 = x_i^0 + \Delta$.) (2) The parties need to use a "committed OPE" protocol, where $\mathcal{P}_2$ commits to the coefficients of its polynomial (such a protocol has not been described yet, but it is not hard to imagine how to implement it using techniques similar to those used for committed OT). (3) $\mathcal{P}_2$ must prove that there are values s, d such that s is committed to in $\mathsf{Com}(\sigma_r)$, d is committed to in $\mathsf{Com}(\Delta)$, and it holds that $P(s+d) = P(s+2d) = \cdots = P(s+\ell d)$. The main challenge in designing this step is that $P(s+d)$ is computed to by multiplying the committed coefficients of P by powers of the value $s+d$. Namely, the proof is about the sum of multiplications of committed values.

6 *m*-Point SPIR

Another application of the HDOT protocol is a new variant of symmetric private information retrieval (SPIR – Symmetric PIR) which we denote as m-point-SPIR. For a definition and discussion of single server PIR and symmetric PIR, see, e.g. [21,5]. In short, a PIR protocol involves a server with a database of N items $x_0, \ldots, x_{N-1}$ and a client who is interested in learning entry x_i of the database. This must be accomplished with $o(N)$ communication, without revealing i to the server, and (in the case of *symmetric* PIR) without revealing to the client anything but x_i.

The m-point-SPIR protocol that we define can be applied if at most m of the items of the server's database have specific values, and all other items have some default value $\bar{x}$. The client must not know whether the value it learns is the default value $\bar{x}$ or one of the unique values. We describe below a couple of applications of m-point-SPIR. The m-point-SPIR functionality is similar to a simpler functionality, where the client learns a *random* value if its input does not match any of the m indices which have specific values. The latter functionality is much simpler to implement (using OPE), as we detail below.

We show a protocol which implements m-point-SPIR with $O(m \log N)$ communication and $O(m \log N)$ computation (the smaller m is, the more efficient the protocol is). Therefore the communication is $o(N)$ as long as $m = o(N/\log N)$.

Another nice property of the m-point-SPIR protocol if that it can be implemented based on the existence of oblivious transfer alone. This property is not known for general SPIR protocols. (Furthermore, it is known that there cannot exist any transparent black-box reduction of PIR to OT [25].)

The m-point-SPIR functionality is defined in the following way. The server has inputs $0 \leq p_1, \ldots, p_m \leq N-1$, which are all distinct, and additional values $\bar{x}, x_{p_1}, \ldots, x_{p_m}$. The client has an input $0 \leq i \leq N-1$. The output of the client is x_{p_j} if there is an index $1 \leq j \leq m$ such that $i = p_j$, or $\bar{x}$ if no such p_j exists.

1-point SPIR. The implementation of 1-point-SPIR is straightforward given our previous protocols. The parties simply execute the protocol $EQ_{x_{p_1},\bar{x}}(i, p_1)$, whose output is x_{p_1} if $i = p_1$, and $\bar{x}$ otherwise. (The EQ protocol is defined in Section 3.1.) The communication overhead is of the order of the length of the index i, namely $O(\log N)$, times the length of the security parameter (i.e., the length of the homomorphic encryption). (This is under the reasonable assumption that the length of the database values (the x values) is in the order of the length of the security parameter; otherwise the communication is $O(\log N \cdot |x|)$.) The computation overhead is $O(\log N)$, and it is composed of $O(\log N)$ homomorphic encryptions and $O(\log\log N)$ OTs.

m-point-SPIR. For the general case of m-point-SPIR, the server first defines m random values $z'_1, \ldots, z'_m$ under the constraint that their exclusive-or is $\bar{x}$. It then defines values $z_1, z_2, \ldots, z_m$ satisfying the constraints $z_1 \oplus z'_2 \oplus \cdots \oplus z'_m = x_1$, $z'_1 \oplus z_2 \oplus z'_3 \oplus \cdots \oplus z'_m = x_2$, up to $z'_1 \oplus \cdots \oplus z'_{m-1} \oplus z_m = x_m$. The parties execute the protocols $EQ_{z_1,z'_1}(i, p_1)$, $EQ_{z_2,z'_2}(i, p_2)$, up to $EQ_{z_m,z'_m}(i, p_m)$. The client then computes the exclusive-or of the m values that it learned in these protocols.

Correctness follows from the fact that if there exists a j coordinate for which $i = p_j$ then the client learns a single z_j value. Otherwise $i \neq p_1, \ldots, p_m$ and the client learns only z'_j values. Therefore the exclusive-or of all the values that the client receives is equal to x_j in the former case, or to $\bar{x}$ in the latter case.

It is easy to verify the security of this protocol (assuming that the parties are semi-honest). Note that the client always performs the same operations and does not recognize whether it learned the value $\bar{x}$ or one of the m special values. The communication overhead is $O(m \log N)$ times the length of the security parameter, and the computation overhead is also $O(m \log N)$. This is therefore a SPIR protocol (with $o(N)$ communication) as long as $m = o(N/\log N)$, and in that case the computation overhead is also $o(N)$. (A "traditional" PIR protocol will have $O(N)$ computation overhead, since it must also process the entries with the default value.)

Basing m-point-SPIR on OT. The EQ protocol (which is essentially the HDOT protocol) is based on using a homomorphic encryption scheme and an oblivious transfer. However, it is easy to see that the usage of homomorphic encryption can be replaced with the usage of oblivious transfer alone (as is done in the HDOT protocol for the malicious case). As a result, m-point-SPIR can be based oblivious transfer alone.

Comparison to other protocols. Our m-point-SPIR protocol can be compared to oblivious polynomial evaluation (OPE), in which the server has an $(m-1)$-degree polynomial P, defined over a field of size at least N, and where the polynomial satisfies $P(p_j) = x_j$ for all $j \in [1, m]$. The client has input $0 \leq j \leq N-1$ and it obliviously computes $P(j)$. The OPE protocol has communication and computation overheads of $O(m)$ field operations, but it has the drawback that for inputs not in $p_1, \ldots, p_m$ the client receives a random output rather than a specific value $\bar{x}$.

The m-point-SPIR protocol can also be compared to PIR protocols of the type of the protocol of Cachin, Micali and Stadler [5] (that protocol is based on the ϕ-hiding assumption rather on general assumptions). These protocols, too, have the property that the server's work depends on the number of items in its database that have non-default values. Namely, it is $O(m)$ if the server has m items in its database, even if the range of the client's input is $[1, N]$. Still, in those protocols the sender is not able to set a "default" value $\bar{x}$ to be returned for all other $N-m$ values of the client's input. Finally, the m-point-SPIR functionality can be implemented using Yao's generic protocol and a circuit of size $O(m \log N)$, and $m \log N$ invocations of OT. The observations in Section 3 comparing the overhead of the HDOT protocol to that of Yao's construction, are relevant in this case, too. We also believe that it is simpler to implement the m-point-SPIR protocol compared to implementing a circuit based solution.

Application I: private matching for cardinality threshold. This is an example where it is important that $\mathcal{P}_1$ receives the default value if no match is found. The scenario involves two parties with private sets of m items, which want to find out if the size of the intersection of the sets is greater than some threshold. The problem was defined in [13] as a variant of the private matching protocol which was the main subject of that paper. The solution there requires the parties to run an OPE for each item x_i of the first party, in which the first party either learns a specific value or a random value, depending on whether x_i is in the set of the second party. The parties then use Yao's protocol to evaluate a circuit whose input is the values learned by $\mathcal{P}_1$, and which computes whether the size of the intersection is greater than the threshold. We can use the m-point-SPIR protocol to replace the OPE: Suppose that $\mathcal{P}_1$'s inputs are $x_1, \ldots, x_n$ and $\mathcal{P}_2$'s inputs are $y_1, \ldots, y_n$. Then for each x_i the parties run an m-point SPIR where $\mathcal{P}_1$ learns α_i if $x_i \in \{y_1, \ldots, y_n\}$, or $\alpha_i + 1$ otherwise, where α is a random number chosen by $\mathcal{P}_2$. We can then ask $\mathcal{P}_1$ to sum the values it learned, and replace Yao's protocol with an OT_1^m , as was done in the *bin*HDOT protocol of Section 4.1. (This was impossible when an OPE was used, since in that case the sum was random if there was even a single item of $\mathcal{P}_1$ which was not in $\mathcal{P}_2$'s set.)

Application II: lottery service As an example of another application of m-point-SPIR, consider a lottery service where the server has a range of tickets, only a few of which are winning tickets. The client uses the protocol to "buy" a ticket, but the client must not know, at least not until some time in the future, whether this is a winning ticket. The server's database contains the prize corresponding to each winning ticket, or the default "no prize" value $\bar{x}$ (which, of course, is associated to most of the tickets). It must be ensured that a client that receives the value $\bar{x}$ cannot identify that this is the default value. The server must not learn which ticket was chosen by the buyer. (A lottery service with many clients must handle many other different issues which we do not describe, but m-point-SPIR seems like a good approach for handling the purchase of tickets.)

References

1. Ben-David, A., Pinkas, B., Nisan, N.: Fairplaymp – a system for secure multi-party computation. In: ACM Conference on Computer and Communications Security—ACM CCS 2008. ACM, New York (2008)
2. Blake, I.F., Kolesnikov, V.: Conditional encrypted mapping and comparing encrypted numbers. In: Crescenzo and Rubin [9], pp. 206–220
3. Bogetoft, P., Damgård, I., Jakobsen, T., Nielsen, K., Pagter, J., Toft, T.: A practical implementation of secure auctions based on multiparty integer computation. In: Crescenzo and Rubin [9], pp. 142–147
4. Boneh, D. (ed.): CRYPTO 2003. LNCS, vol. 2729. Springer, Heidelberg (2003)
5. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999)
6. Camenisch, J., Neven, G., Shelat, A.: Simulatable adaptive oblivious transfer. In: Naor [26], pp. 573–590
7. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh [4], pp. 126–144
8. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptology 13(1), 143–202 (2000)
9. Di Crescenzo, G., Rubin, A. (eds.): FC 2006. LNCS, vol. 4107. Springer, Heidelberg (2006)
10. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. In: Advances in Cryptology - Crypto 1982, pp. 205–210 (1982)
11. Fagin, R., Naor, M., Winkler, P.: Comparing information without leaking it. Communications of the ACM 39(5), 77–85 (1996)
12. Feigenbaum, J., Ishai, Y., Malkin, T., Nissim, K., Strauss, M.J., Wright, R.N.: Secure multiparty computation of approximations. ACM Transactions on Algorithms 2(3), 435–472 (2006)
13. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)
14. Goethals, B., Laur, S., Lipmaa, H., Mielikäinen, T.: On private scalar product computation for privacy-preserving data mining. In: Park, C.-s., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 104–120. Springer, Heidelberg (2005)
15. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press, New York (2004)
16. Green, M., Hohenberger, S.: Blind identity-based encryption and simulatable oblivious transfer. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 265–282. Springer, Heidelberg (2007)
17. Hazay, C., Lindell, Y.: Efficient oblivious polynomial evaluation and transfer with simulation-based security (manuscript) (2008)

18. Indyk, P., Woodruff, D.P.: Polylogarithmic private approximations and efficient matching. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 245–264. Springer, Heidelberg (2006)
19. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner [33], pp. 572–591
20. Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. In: Naor [26], pp. 97–114
21. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: single database, computationally-private information retrieval. In: FOCS 1997, pp. 364–373 (1997)
22. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor [26], pp. 52–78
23. Lindell, Y., Pinkas, B., Smart, N.P.: Implementing two-party computation efficiently with security against malicious adversaries. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 2–20. Springer, Heidelberg (2008)
24. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: USENIX Security Symposium, pp. 287–302. USENIX (2004)
25. Meier, R., Przydatek, B.: On robust combiners for private information retrieval and other primitives. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 555–569. Springer, Heidelberg (2006)
26. Naor, M. (ed.): EUROCRYPT 2007. LNCS, vol. 4515. Springer, Heidelberg (2007)
27. Naor, M., Nissim, K.: Communication preserving protocols for secure function evaluation. In: STOC, pp. 590–599 (2001)
28. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: STOC 1999, pp. 245–254. ACM Press, New York (1999)
29. Naor, M., Pinkas, B.: Computationally secure oblivious transfer. J. Cryptology 18(1), 1–35 (2005)
30. 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)
31. Paillier, P.: Trapdooring discrete logarithms on elliptic curves over rings. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 573–584. Springer, Heidelberg (2000)
32. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner [33], pp. 554–571
33. Wagner, D. (ed.): CRYPTO 2008. LNCS, vol. 5157. Springer, Heidelberg (2008)
34. Wright, R., Yang, Z.: Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In: Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 713–718. ACM Press, New York (2004)
35. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: FOCS, pp. 162–167. IEEE, Los Alamitos (1986)

Efficient Robust Private Set Intersection

Dana Dachman-Soled[1], Tal Malkin[1], Mariana Raykova[1], and Moti Yung[2]

[1] Columbia University
{dglasner,tal,mariana}@cs.columbia.edu
[2] Columbia University and Google Inc.
moti@cs.columbia.edu

Abstract. Computing Set Intersection privately and efficiently between two mutually mistrusting parties is an important basic procedure in the area of private data mining. Assuring robustness, namely, coping with potentially arbitrarily misbehaving (i.e., malicious) parties, while retaining protocol efficiency (rather than employing costly generic techniques) is an open problem. In this work the first solution to this problem is presented.

Keywords: Set Intersection, Secure Two-party Computation, Cryptographic Protocols, Privacy Preserving Data Mining.

1 Introduction

Constructing an efficient, robust two-party protocol for computing set intersection that is secure and realizable given current encryption methods is an open question first introduced in the work of Freedman, Nissim and Pinkas [9]. Here we solve this problem and present a protocol that allows two mutually distrustful parties holding private inputs to compute the intersection of their inputs without revealing any additional information. We prove the security of our protocol in the standard Ideal/Real Model. The Set Intersection primitive is widely used in the area of privacy preserving data mining ([19]); the prototypical application involve secure sharing of information in areas like personal health and finance.

Although generic robust methods (c.f., [20]) based on Yao's general two-party computations [25] are sufficient for computing any two-party functionality, here we are after efficient methods. Since the size of the naive circuit needed to compute Set Intersection is at least $\Omega(m \cdot n)$ (where n is the input size of the party that receives output and m is the input size of the other party) any generic construction for semi-honest two-party computation will have communication complexity $\Omega(m \cdot n)$, even without robustness. In contrast, our protocol's communication complexity is $O(mk^2 \log^2 n + kn)$ ciphertexts, where k is a security parameter (i.e. logarithm of the size of the field, where we allow sets that are arbitrary but are representable in this field). Additional properties of our solution are worth mentioning: First, the number of exponentiations needed by our protocol increases only by a poly-logarithmic (i.e. a $k^2 \log^2 n$) factor in comparison to the number of exponentiations required by the semi-honest protocol of [9]

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 125–142, 2009.

over domains as above. Secondly, our construction is fully-black box assuming the existence of a homomorphic encryption scheme. Finally, the encryption is only required to possess a few natural properties, which are discussed in the following section (and satisfied by known homomorphic encryption schemes, e.g., based on DDH).

Our Methodology and Techniques. Our starting point is the semi-honest protocol of [9] that computes set intersection via polynomial evaluation. In this protocol, the Server must evaluate an encrypted polynomial of degree n on each of his inputs and send the (encrypted) results back to the Client. In the malicious adversary case, though, to achieve security, the Client must be able to verify that the Server evaluated the polynomial honestly with respect to its input. To ensure this, we use techniques that add redundancy to the representation of the inputs (this is motivated by techniques in Choi et. al [6]). More specifically, we employ a Server that shares its input via a Shamir secret-sharing [23] threshold scheme using a degree k polynomial, where k is the security parameter, and then commits to shares of its input. Note that a Shamir secret-sharing can also be viewed as a Reed-Solomon encoding of the input. What we would like to do next is have the server evaluate the encrypted polynomial on each share of its input and send the resulting shares to the Client. Note that due to the fact that polynomials are closed under composition, the above yields a valid-secret sharing (and a valid Reed-Solomon encoding) of the output value. However, the resulting polynomial is now of degree $n \cdot k$, and so we need at least $n \cdot k + 1$ shares to recover the secret. To improve our efficiency, we apply input-preprocessing technique that allows us to reduce the degree of the output polynomial to $d = k(\lfloor \log n \rfloor + 1) + k$, and thus we need only $O(k(\lfloor \log n \rfloor + 1))$ shares in order to recover the shared value.

Next, we ensure that the Server acted honestly for a large-fraction of the shares by executing a cut-and-choose protocol that forces the Server to open k random shares for each committed input value, thus allowing the Client to verify that the corresponding output share was computed correctly. Due to the information-theoretic security of the secret-sharing scheme, no information about the input is leaked by opening these shares. Additionally, the Client checks that all the output shares he received indeed lie on the same polynomial of degree d. Due to the large distance between codewords in a Reed-Solomon code this ensures that, in fact, *all* the shares were computed exactly correctly. Finally, the Client reconstructs the secret, which is now guaranteed to be consistent with the Server's inputs. Note that in the two-party case, we only need to either complete the computations if the other party acts honestly, or detect cheating. This allows us to use Lagrange interpolation and a consistency check as an error detection code, rather than error-correction (implied by techniques such as Berlkamp-Welch). As is noted in the sequel, this is important to the realization of the encryption schemes, since the interaction of the algebra of secret sharing methods and the algebra of concrete encryption schemes is a subtle issue (not treated in earlier work). We ensure that every algebraic operation used in our protocol is realizable given a concrete encryption scheme.

Related Work. Multiple papers address the problem of secure set intersection and suggest various solutions [1,9,18,12]. (We remark that, in addition, several works deal with variants of set intersection such as the private equality test for input sets of size one [8,21,3,14] or the problem of disjointness that asks whether the intersection of two sets is empty ([17])). However, none of the protocols for set intersection that have been suggested thus far are secure in the scenario of arbitrarily malicious adversaries and arbitrary input domains. Freedman et. al([9]) present a protocol claimed (without details or proofs) to be secure in the presence of a malicious Client in the standard Ideal/Real model, and secure in the presence of a malicious Server only in the Random Oracle model. Hazay and Lindell ([12]), in turn, adopt a different approach based on secure pseudorandom function evaluation that does not use random oracles but they only achieve security against a malicious Client (and semi-honest Server), or security against two *covert* parties, where covert is a new non-standard model that is stronger than semi-honest, but weaker than malicious. Recently (and independently of our work), Jarecki and Liu ([15]) extend the approach of [12] to provide a protocol secure against two malicious parties, when the input sets are chosen from a polynomial-sized domain and based on the Decisional q-Diffie-Hellman Inversion Assumption. We also note the work of Kissner and Song ([18]) which presents multi-party protocols that are secure in the presence of semi-honest adversaries for several set operations including Set Intersection. Additionally, they briefly address achieving security in the presence of malicious adversaries, but their method relies on inefficient generic zero-knowledge proofs. Also independently, Camenisch and Zaverucha ([4]) extend the protocol of [9] in a different direction where they assume the presence of a certifying third party that signs the input sets of the two participants. This provides guarantees that the set intersection functionality is computed correctly with respect to the signature certified input sets in the presence of malicious adversaries.

Organization. In section 2 we present definitions and known building blocks, while in Section 3 we present our protocol steps and protocol. In Section 4 we present some intuition for the proof of security, and discuss our complexity.

2 Definitions and Building Block Protocols

We use a standard simulation-based definition of security from [5], and follow the definitions of zero knowledge proofs of knowledge and commitment schemes from [10]. We denote Com_B a perfectly binding commitment scheme and Com_H a perfectly hiding commitment scheme.

We follow the standard definitions of semantically-secure encryption schemes and homomorphic encryption schemes given in [16]. We assume the plaintexts of the semantically-secure encryption scheme ENC are elements of a finite group P with group operation '+' and that the ciphertexts are elements of a finite group C with group operation '$\cdot$'. Since ENC is a homomorphism from P to C, the homomorphic property of an encryption scheme ENC can be stated as follows:

Property 1 (Homomorphic Encryption).

$$\mathrm{ENC}(X, r_1)\cdot\mathrm{ENC}(Y, r_2) = \mathrm{ENC}(X+Y, r) \qquad (\mathrm{ENC}(X, r_3))^{\lambda} = \mathrm{ENC}(\lambda\cdot X, r').$$

We will also require that r can be computed in polynomial-time given r_1, r_2, X, Y, r' can be computed in polynomial time given r_3, X, λ, and that r, r' are uniformly distributed when r_1, r_2, r_3 are (so the encryptions after applying a homomorphic operation are distributed as random encryption). It turns out that known homomorphic encryption schemes typically satisfy the above requirements, and actually possess the following, stronger, property:

Property 2.

$$\mathrm{ENC}(X, r_1)\cdot\mathrm{ENC}(Y, r_2) = \mathrm{ENC}(X+Y, r_1+r_2);\ (\mathrm{ENC}(X, r_3))^{\lambda} = \mathrm{ENC}(\lambda\cdot X, \lambda\cdot r_3).$$

This property is satisfied by most known homomorphic encryption schemes, such as Paillier [22], ElGamal [7], and Goldwasser-Micali [11] encryption schemes.

We also present the Additive El-Gamal Encryption scheme, which we will use to concretely instantiate our protocol:

Definition 1 (Additive El Gamal Encryption Scheme: AEG_{enc}).

- GEN*: on input* 1^n *generate* (G, q, g) *where* q *is prime,* G *is a cyclic group of order* q *and* g *is a generator. Then choose a random* $x \leftarrow Z_q$ *and compute* $h = g^x$*. The public key is* $\langle G, q, g, h\rangle$ *and the private key is* $\langle G, q, g, x\rangle$*.*
- ENC*: on input a public key* $pk = \langle G, q, g, h\rangle$ *and a message* $m \in Z_q$*, choose a random* $y \leftarrow Z_q$ *and output the ciphertext*

$$\langle g^y, h^y \cdot g^m\rangle$$

- DEC*: on input a private key* $sk = \langle G, q, g, x\rangle$*and a ciphertext* $\langle c_1, c_2\rangle$*, output*

$$g^m = c_2/c_1^x$$

Unlike regular Additive El Gamal decryption, here we can recover g^m and not necessarily know m. However, this will be sufficient for our application and we are able to handle plaintexts that come from a large domain.

Now we proceed to define several auxiliary protocols that will be used in our main protocols.

2.1 Homomorphic Encryption Proof of Knowledge

This protocol will be used by both the Server and Client when a party P_0 sends to a party P_1 a public key pk and several values encrypted under ENC_{pk}. In the malicious case, we require P_0 to prove that he knows the corresponding plain text values and randomness and additionally that the encrypted plaintexts are "valid" (i.e. belong to a particular language). This protocol is similar to the polynomial time provers in [13].

If P_1 is behaving honestly, its input should be a member of the language L whose membership can be determined in polynomial time and is closed under addition and subtraction. The NP-language L', is defined as follows:

$$L' = \{\overline{C} = (pk1, c_1, \ldots, c_\alpha) \mid \; c_i = \text{ENC}_{pk1}(x_i; r_i), \text{ for some } x_i, r_i, 1 \leq i \leq \alpha, \\ \text{and } (x_1, \ldots, x_\alpha) \in L\}$$

Homomorphic Encryption Proof of Knowledge and Plaintext Verification (HEPKPV) Protocol: Π_{POK}

Input: $P_0 \leftarrow \overline{C} = (pk1, c_1, \ldots, c_\alpha)$, $(x_1, \ldots x_\alpha) \in L$, $(r_1, \ldots, r_\alpha)$ where $c_i = \text{ENC}_{pk1}(x_i; r_i)$ for $1 \leq i \leq \alpha$;
$P_1 \leftarrow \overline{C} = (pk1, c_1, \ldots, c_\alpha)$
Output: P_1 outputs *Accept* if $\overline{C} \in L'$, and *Reject* otherwise.

1. P_0 chooses k random vectors $(e_{11}, \ldots e_{1\alpha}), \ldots, (e_{k1}, \ldots, e_{k\alpha})$ such that for $1 \leq i \leq k$, $(e_{i1}, \ldots, e_{i\alpha}) \in L$. and another k vectors $(r_{11}, \ldots r_{1\alpha}), \ldots, (r_{k1}, \ldots, r_{k\alpha})$ of random numbers.
2. P_0 computes the encryptions $(c_{i1}, \cdots, c_{i\alpha}) = (\text{ENC}(e_{i1}, r_{i1}), \ldots, \text{ENC}(e_{i\alpha}, r_{i\alpha}))$ for $1 \leq i \leq k$ and sends them to the Server.
3. P_1 chooses a sequence of k bits $b'_1 \ldots b'_k$ and sends to P_1 a commitment to those bits: $Com_H(b'_1 \ldots b'_k)$, along with the public parameters for the commitment scheme.
4. P_0 chooses a sequence of k bits $b''_1 \ldots b''_k$ and sends to P_0 a commitment to those bits: $Com_B(b''_1 \ldots b''_k)$, along with the public parameters for the commitment scheme.
5. P_0 and P_1 decommit the value $b''_1 \ldots b''_k$ and $b'_1 \ldots b'_k$, respectively.
6. P_0, P_1 verify that the bits received correspond to the commitments that were sent. If the check fails, they abort the protocol. Otherwise both P_0 and P_1 compute $b_1 \ldots b_k = b'_1 \ldots b'_k$ **XOR** $b''_1 \ldots b''_k$.
7. For each $1 \leq i \leq k$:
   (a) if $b_i = 0$, P_0 sends to P_1 $\overline{M} = (e_{i1}, \cdots e_{i\alpha})$ and $\overline{R} = (r_{i1}, \cdots r_{i\alpha})$;
   (b) if $b_i = 1$, P_0 sends to P_1 $\overline{M} = (x_1 + e_{i1}, \cdots, x_n + e_{i\alpha})$ and $\overline{R} = (r_1 + r_{i1}, \cdots, r_\alpha + r_{i\alpha})$.
8. For each $1 \leq i \leq k$:
   (a) if $b_i = 0$, P_1 verifies that $(c_{i1}, \cdots, c_{i\alpha}) = (\text{ENC}(e_{i1}, r_{i1}), \cdots \text{ENC}(x_{i\alpha}, r_{i\alpha}))$;
   (b) if $b_i = 1$, P_1 verifies that $(c_1 c_{i1}, \cdots c_\alpha c_{i\alpha}) = (\text{ENC}(x_1 + e_{i1}, r_1 + r_{i1}), \cdots, \text{ENC}(x_\alpha + e_{i\alpha}, r_\alpha + r_{i\alpha}))$.
9. P_1 verifies that $(\overline{M}) \in L$.
10. If any of the verifications steps of P_1 fail, abort the protocol. Otherwise, accept.

Lemma 1. *Assume that $Hom_{enc} = (Gen, Enc, Dec)$ is a CPA-secure homomorphic encryption scheme, Com_H is a perfectly hiding commitment scheme, and Com_B is a perfectly binding commitment scheme. Then protocol Π_{POK} is a Zero Knowledge Proof of Knowledge for L'.*

See full version for proof.

Now we define several languages that we will use in the main protocols in the context of the above HEPKPV protocol:

- Language consisting of points that lie on some polynomial of degree ℓ

$$\begin{aligned} L_{poly}(t,u,\ell) = \{m_{i,j} \mid 1 \le i \le t,\ 1 \le j \le u;& \\ \text{for each } j \text{ the points } ((1,m_{1,j}),\ldots,(t,m_{t,j}))& \\ \text{lie on a polynomial of degree } \ell\}.& \end{aligned}$$

- Language consisting of points that lie on some polynomial of degree ℓ that has zero free coefficient

$$\begin{aligned} L_{poly,0}(t,u,\ell) = \{m_{i,j} \mid 1 \le i \le t,\ 1 \le j \le u;& \\ \text{for each } j \text{ the points } ((1,m_{1,j}),\ldots,(t,m_{t,j}))& \\ \text{lie on a polynomial } P_j \text{ of degree } \ell \text{ and } P_j(0) = 0\}.& \end{aligned}$$

- Language consisting of points that lie on some polynomial of degree ℓ that has free coefficient equal to m'_j.

$$\begin{aligned} L_{eq}(t,u,\ell) = \{m_{i,j}, m'_j \mid 1 \le i \le t; 1 \le j \le u,& \\ \text{for each } j \text{ the points } ((1,m_{1,j}),\ldots,(t,m_{t,j}))& \\ \text{lie on a polynomial } P_j \text{ of degree } \ell, \text{ where } P_j(0) = m'_j\}& \end{aligned}$$

- The following language consists of several tuple of pairs of the form $(m_{i,j}, m'_{i,j})$. For each i the points $(1,m_{i,1}),\ldots,(t,m_{i,t})$ lie on a polynomial P_i of degree ℓ and the points $(1,m'_{i,1}),\ldots,(t,m'_{i,t})$ lie on a polynomial R_i of degree 2ℓ and additionally, for each i, $P_{i+1}(0) = R_i(0)$.

$$\begin{aligned} L_{sq}(t,u,\ell) = \{(m_{i,j}, m'_{i,j}) \mid 1 \le i \le u,\ 1 \le j \le t;\ \text{for all } i,& \\ \text{the points } ((1,m_{i,1}),\ldots,(t,m_{i,t})) \text{ lie on } P_i \text{ of degree } \ell;& \\ \text{the points } ((1,m'_{i,1}),\ldots,(t,m'_{i,t})) \text{ lie on } R_i \text{ of degree } 2\ell;& \\ \text{and for } 1 \le i \le u-1, P_{i+1}(0) = R_i(0)\}& \end{aligned}$$

Membership in all of the above languages can be determined in polynomial time. Also these languages are closed under addition and can be used in the context of the HEPKPV protocol.

2.2 Coin Tossing

The following protocol is run by the Server S and Client C in order to select a random number within a given range $[0, s-1]$ known to both of them. At the end of the protocol both parties obtain the same random number. The private input of the two parties is $(\perp, \perp)$ and the output to each party is $(rand, rand)$, where $rand$ is a uniformly random number chosen from $[0, s-1]$.

1. S chooses a random value $R' \in [0, s-1]$ and sends a commitment $C_1 = Com_H(R')$ to P_1.
2. C chooses a random value $R'' \in [0, s-1]$ and sends a commitment $C_2 = Com_B(R'')$ to P_0.
3. C opens the commitments C_2.
4. S opens the commitment C_1.
5. The two parties output $R = R' + R'' \mod s$.

We note that both statistically hiding and statistically binding commitments can be constructed using a homomorphic encryption scheme.

Lemma 2. *Assume that Com_H is a statistically hiding commitment scheme and Com_B is a perfectly binding commitment scheme. Then protocol Π_{Coin} is simulatable for Malicious C and Honest S.*

See full version for proof.

3 Set Intersection Protocol

We now describe the setting for the Set Intersection protocol. There are two participants in the protocol: Client, C and Server, S. The Client has an input set $X, |X| = n$ of size at most $n \leq max_c$ and the Server has an input set $Y, |Y| = m$ of size at most $m \leq max_s$. Both parties know a homomorphic encryption scheme $Hom_{enc} = (\text{GEN}, \text{ENC}, \text{DEC})$. Further the Client and the Server choose a security parameter k. The goal of the protocol is that the Client learns the intersection of their sets: $X \bigcap Y$ and nothing else while the Server learns nothing. Now if the pair (K_c, K_s) represents knowledge of the Client (K_c) and the Server (K_s), the input and output of the set intersection protocol can be summarized as follows:

$$\begin{matrix} (\{X, max_c, max_s, Hom_{enc}, k\}, \\ \{Y, max_s, max_c, Hom_{enc}, k\}) \end{matrix} \rightarrow \begin{cases} (X \bigcap Y, \perp), & \text{if } |X| \leq max_c,\ |Y| \leq max_s \\ \quad (\perp, \perp) & \text{otherwise} \end{cases}$$

Our idea starts with the approach of [9]: the Client constructs a polynomial P of degree n over a finite field such that $P(x) = 0$ if and only if $x \in X$. The Client encrypts the coefficients of P using a homomorphic encryption scheme and sends them to the Server. Due to the homomorphic properties of the encryption scheme, the Server is now able to evaluate the polynomial at each of its inputs. Thus, for $1 \leq \ell \leq m$, the Server sends the encryption of the following output back to the Client: $r_j \cdot P(y_j) + y_j$, where r_j is chosen randomly. Thus, we have that if $y_j \in X \cap Y$ then the Client receives y_j. If $y_j \notin X \cap Y$ then the Client receives a random value.

Before presenting our main protocol, we define three protocols that are used as building blocks for the main protocol. They implement the two main ideas that we use to achieve security against malicious parties.

3.1 Input Sharing via Enhanced Shamir Scheme

In the set intersection protocol we "share" function evaluation by secret sharing the arguments of the function and evaluating the function on corresponding shares in order to obtain shares of the final value of the function. We use Shamir's secret sharing ([23]) but for the purposes of efficiency we apply the following further transformation on the inputs.

Let f be a polynomial of degree n over a single variable: $f = a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$. For a given z and r, we would like to obtain shares of the result $g(z,r) = r \cdot f(z) + z$ by secret-sharing the inputs z and r. For this purpose we choose random polynomials P_z, P_r of degree k for such that $P_z(0) = z$ and $P_r(0) = r$ and evaluate g on m shares of the input to obtain $g(P_z(1), P_r(1)), \dots g(P_z(m), P_r(m))$. We now define a new single variable polynomial $g'(i) = g(P_z(i), P_r(i))$. Note that the degree of g' is $n \cdot k + k$ and that $g'(0) = g(P_z(0), P_r(0)) = g(z,r)$ and thus given $g(P_z(1), P_r(1)), \dots, g(P_z(m), P_r(m)) = g'(1), \dots, g'(m)$ we can reconstruct $g'(0) = g(z,r)$ when $m \geq n \cdot k + k$. This means that the number of shares needed is at least $n \cdot k + k$.

We extend the above idea further in order to decrease the degree of the final sharing polynomial of the result. For a given z and r, we obtain shares of the result $g(z,r)$ in the following way. For $0 \leq \ell \leq \lfloor \log n \rfloor$ we secret share the value z^{2^ℓ} using a random polynomial $P_{z^{2^\ell}}$, of degree k, such that $P_{z^{2^\ell}}(0) = z^{2^\ell}$. Let $s[i]$ indicate the $i^t h$ bit of a number s. We now define a new polynomial $g''(i) = P_z(i) + P_r(i) \cdot \Sigma_{s=1}^{n} a_s \cdot \Pi_{\ell=1}^{\lfloor \log n \rfloor + 1} (P_{z^{2^\ell}}(i))^{s[\ell]}$. Note that $g''(0) = g(z,r)$ and that g'' has degree $(\lfloor \log n \rfloor + 1)k + k$. We have thus drastically reduced the number of shares necessary to recover $g''(0) = g(z,r)$.

The above idea for function transformation guarantees correct evaluation of shares of the functional value if the party is following the protocol honestly. In the malicious case a party needs to prove that the sharing functions that it is using for the new arguments $z, z^2, \dots, z^{2^\ell}$ have been constructed correctly. The following protocol allows a party to generate shares of an input z using the preprocessing idea and then prove that these shares were computed correctly without revealing any information about z.

Efficient Preprocessing of Input:

1. For each $y_j \in Y$, $1 \leq j \leq m$ S chooses a random polynomial P_{y_j} of degree k such that $P_{y_j}(0) = y_j$, and computes shares of the form $P_{y_j}(i)$ for $1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)$ and the corresponding encryptions $\mathrm{ENC}_{pk}(P_{y_j}(i))$.
2. For each y_j, $1 \leq \ell \leq m$ S and C run the HEPKPV protocol with $L_{poly}(m,n,k)$ $= \{m_{i,j} = P_{y_j}(i)\}$ in order for S to prove the correctness of his sharing.
3. For each y_j, for $\ell = 0$ to $\lfloor \log n \rfloor$, for $i = 0$ to $10k(\lfloor \log n \rfloor + 1)$, S computes the following:
   - Local Computation on Shares: a polynomial $P^2_{y_j^{2^\ell}}$ of degree $2k$ such that $P^2_{y_j^{2^\ell}}(i) = (P_{y_j^{2^\ell}}(i))^2$
   - Degree Reduction Step: a random polynomial $P_{y_j^{2^{\ell+1}}}$ of degree k such that $P_{y_j^{2^{\ell+1}}}(0) = P^2_{y_j^{2^\ell}}(0)$.

4. For each y_j, for $\ell = 0$ to $\lfloor \log n \rfloor$, for $i = 1$ to $10k(\lfloor \log n \rfloor + 1)$, S computes the following commitments:
   - New input shares: $\mathrm{ENC}_{pk}(P_{y_j^{2^{\ell+1}}}(i))$ and
   - Intermediate shares: $\mathrm{ENC}_{pk}(P^2_{y_j^{2^\ell}}(i))$

   and sends those commitments to C

We now describe how C verifies S's computation of its new shares.

Let J be the ordered set of all elements of $\{0,1\}^{10k(\lfloor \log n \rfloor + 1)}$ that contain exactly k ones. Note that given R, an index of a string in the set J, we can efficiently reconstruct the Rth string, j_R. Let $J_R = \{i | j_R[i] = 1\}$, where $j_R[i]$ denotes the ith position of the string j_R.

Preprocessing Verification:

Common Inputs: The commitments: $[C_{j,\ell,i}]_{1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor + 1, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[C^2_{j,\ell,i}]_{1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, and a number $R \in [|J|]$ chosen using the Coin-Tossing protocol after S committed to its inputs.

Private Inputs of S: Decommitments to the above values.

1. For all $i \in J_R, 1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor$ S opens the commitment $C^2_{j,\ell+1,i}$ to $C'^2_{j,\ell+1,i}$, $C_{j,\ell,i}$ to $C'_{j,\ell,i}$.
2. For all $i \in J_R, 1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor$ C checks that $C'^2_{j,\ell+1,i} = (C'_{j,\ell,i})^2$.
3. S and C run the HEPKPV protocol with S's private inputs, the common commitment inputs and language $L_{sq}(\lfloor \log n \rfloor, 10k(\lfloor \log n \rfloor + 1), m, k) = \{m_{\ell,i,j} = P_{y_j^{2^\ell}}(i), m'_{\ell,i,j} = P^2_{y_j^{2^\ell}}(i)\}$

In the first step of the above protocol S first proves that for all $y_j \in Y, 0 \le \ell \le \lfloor \log n \rfloor$ he has computed correctly $P^2_{y_j^{2^\ell}}(i)$ for at least a .9-fraction of the shares correctly. In the second step of the protocol S proves that for all $y_j \in Y, 0 \le \ell \le \lfloor \log n \rfloor$ he has computed correctly the new sharing polynomials for the values $y_j^{2^{\ell+1}}$ and that both $P_{y_j^{2^\ell}}$ and $P^2_{y_j^{2^\ell}}$ are polynomials. Since any 2 polynomials of degree at most $2k$ must disagree on at least a .8-fraction of the shares, the combination of the above two statements implies that with probability at least $1 - m \cdot (\lfloor n \rfloor + 2)^2 \cdot (1/2^k + .9^k)$, all the sharings were, in fact, computed exactly correctly. For detailed analysis of the above intuition, see the proof sketch in section 4 and the full version.

3.2 Cut-and-Choose on Computations on Input Shares

Common Input: The encryptions: $b_{n+1}, \ldots, b_0$, The commitments: $[M_{i,j,\ell}]_{1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[R_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[0_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[C_{i,j}]_{1 \le \ell \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, and a number $R \in [|J|]$ chosen using the Coin-Tossing protocol after S committed to the above.

Private input of S:Decommitments to$[M_{i,j,\ell}]_{1 \le j \le max_S, 0 \le \ell \le \lfloor \log n \rfloor, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[R_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[Z_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, and the values $[r_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$.

We use the cut-and-choose technique to prove the correctness of evaluation of a specific function on committed inputs $[M_{i,j,\ell}], [R_{i,j}], [Z_{i,j}]$ that results in the commited outputs $[C_{i,j}]$. The function we use is:

$$C_{i,j} = \text{ENC}(0; r_{i,j}) \cdot \text{ENC}_{pk1}(Z'_{i,j}; 0) \cdot \text{ENC}_{pk1}(M'_{i,j,0}; 0) \cdot \left(\varPi_{s=0}^{n} b_s^{\varPi_{\ell=0}^{\lfloor \log n \rfloor} (M'_{i,j,\ell})^{s[\ell]}} \right)^{R'_{i,j}}$$

where $s[\ell]$ denotes the ℓ^{th} bit of s.

We will explain why this is the function we need in the next section.

The steps of the protocols are the following:

1. For each $i \in J_R, 1 \leq j \leq m, 0 \leq \ell \leq \lfloor \log n \rfloor$ S opens the commitments $M_{i,j,\ell}, R_{i,j}, Z_{i,j}$ to $M'_{i,j,\ell}, R'_{i,j}, Z'_{i,j}$ and produces the random value $r_{i,j}$.
2. For $i \in J_R, 1 \leq j \leq m$, C verifies the following: $C_{i,j} =$
$$\text{ENC}_{pk1}(0; r_{i,j}) \cdot \text{ENC}_{pk1}(Z'_{i,j}; 0) \cdot \text{ENC}_{pk1}(M'_{i,j,0}; 0) \cdot \left(\varPi_{s=0}^{n} b_s^{\varPi_{\ell=0}^{\lfloor \log n \rfloor} (M'_{i,j,\ell})^{s[\ell]}} \right)^{R'_{i,j}}$$
3. If any of these verifications fail, C outputs Reject. Otherwise, C outputs Accept.

3.3 Reconstruction and Set Membership Test Protocol

We describe here how the Client reconstructs and checks whether Server's input y_j is in his input set X and consequently in the intersection set using the output shares $[C_{i,j}]_{1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)}$.

1. The Client decrypts the output shares $[C_{i,j}]_{1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)}$ to obtain plaintexts $[C'_{i,j}]_{1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)}$.
2. The Client uses the points $(1, C'_{1,j}), \ldots, (k + k(\lfloor \log n \rfloor + 1), C'_{k+k(\lfloor \log n \rfloor + 1),j}$ and the Lagrange interpolation polynomial to check that for $1 \leq i \leq 10k$ $(\lfloor \log n \rfloor + 1)$
$$C'_{i,j} = L_j(i) = \varSigma_{v=1}^{1+k+k(\lfloor \log n \rfloor + 1)} C'_{j,v} \ell_v(i)$$
where $\ell_v(x) = \varPi_{w=1, w \neq v}^{1+k+k(\lfloor \log n \rfloor + 1)} \frac{x-w}{v-w}$. Otherwise, abort.
3. The Client reconstructs the shared value:
$$C'_{0,j} = L_j(0) = \varSigma_{v=1}^{1+k+k(\lfloor \log n \rfloor + 1)} C'_{j,v} \ell_v(0)$$
and checks whether $C'_{0,j} = x$ for some $x \in X$. If it does, output x.

In the following, we give a concrete implementation of the Reconstruction Protocol using additive El Gamal encryption. We note the following subtlety due to the interaction of the algebraic properties needed to realize the protocol and the properties of the El Gamal encryption scheme. Due to the algebraic properties of the encryption scheme, we are able to compute the Lagrange interpolation polynomial and thus detect errors; however, we cannot run the Berlekamp-Welch algorithm to correct the errors in the codeword. This is due to the fact that the Client can only obtain pairs of the form $(i, g^{m_{i,j}})$ and we are interested in reconstructing a polynomial such that $P(i) = m_{i,j}$ (for a large fraction of i's). The Berlekamp-Welch algorithm requires us to solve a system of linear equations, which we do not know how to do efficiently when we know only $g^{m_{i,j}}$ and not $m_{i,j}$ itself (This issue was ignored in earlier work).

Reconstruction and Set Membership Test via Additive El Gamal Encryption

1. The Client decrypts the output shares $[C_{i,j}]_{1\leq i\leq 10k(\lfloor\log n\rfloor+1)}$ to obtain plaintexts $[g^{m_{i,j}}]_{1\leq i\leq 10k(\lfloor\log n\rfloor+1)}$.
2. The Client uses the points $(1, g^{m_{1,j}}), \ldots, (1 + k + k(\lfloor\log n\rfloor + 1)$, $g^{m_{k+k(\lfloor\log n\rfloor+1),j}}$ and the Lagrange interpolation polynomial to check that for $1 \leq i \leq 10k(\lfloor\log n\rfloor + 1)$
$$g^{m_{i,j}} = L_j(i) = \Pi_{v=1}^{1+k+k(\lfloor\log n\rfloor+1)}(g^{m_{j,v}})^{\ell_v(i)}$$
where $\ell_v(x) = \Pi_{w=1,w\neq v}^{1+k+k(\lfloor\log n\rfloor+1)}\frac{x-w}{v-w}$. Otherwise, abort.
3. The Client reconstructs the shared value:
$$g^{m_{0,j}} = L(0) = \Pi_{v=1}^{1+k+k(\lfloor\log n\rfloor+1)}(g^{m_{j,v}})^{\ell_v(0)}$$
and checks whether $g^{m_{0,j}} = g^x$ for some $x \in X$. If it does, output x.

3.4 The Full Protocol

We start with an overview description of the main steps in the protocol, followed by the detailed specification of our set intersection protocol.

1. The Client runs GEN(1^k) to obtain a secret key sk and a public key pk for Hom_{enc} and sends pk to the Server.
2. The Client computes a polynomial $P(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ of degree the size of his input n over a finite field such that $P(x) = 0$ if and only if $x \in X$.
3. The Client encrypts the coefficients of P, $b_i = \text{ENC}(a_i)$ and sends them to the Server.
4. For each $y_j \in Y$ S chooses a random value r_j and constructs the function
$$\begin{aligned}F(y_j) &= \text{ENC}_{pk_1}(r_j \cdot (y_j) + y_j + 0) = \\ &= \text{ENC}_{pk_1}(0) \cdot \text{ENC}_{pk_1}(y_j) \cdot (\prod_{s=0}^{n}(\text{ENC}_{pk_1}(a_s))^{y_j^s})^{r_j} = \\ &= \text{ENC}_{pk_1}(0) \cdot \text{ENC}_{pk_1}(y_j) \cdot (\prod_{s=0}^{n}(b_s)^{y_j^s})^{r_j}\end{aligned}$$
The above function has the property that it maps the values in the intersection set of the two parties to themselves and values not in the intersection to random numbers.
5. The Server replaces each of its inputs y_j with new variables $c_\ell = y_j^{2^\ell}$ for $0 \leq \ell \leq \lfloor\log n - 1\rfloor$ and transforms the above function to
$$F(y_j) = \text{ENC}_{pk_1}(0) \cdot \text{ENC}_{pk_1}(y_j) \cdot \left(\prod_{s=0}^{n}(b_s)^{\Pi_{\ell=0}^{\lfloor\log n\rfloor}(y_j^{2^\ell})^{s[\ell]}}\right)^{r_j}$$

Note: The exponent of each b_s is y_j^s , however, in the form where s is written in binary and $s[1], ..., s[\lfloor \log n \rfloor + 1]$ are its binary digits and the power of y_j for each digit is substituted with the corresponding new variable from the efficient preprocessing of Servers inputs.

6. The Server shares each of his input $y \in Y$ with polynomial P_{y_j} and each of the random values r_j with a polynomial P_{r_j}.
7. Additionally the Server computes m random polynomials $P_{0,j}$ that have constant coefficient zero. These are used to "rerandomize" the output shares so that they give no information about the input.
8. Using all of the above shares and a random $r_{i,j}$ (to "rerandomize" the encryption) the Server computes shares of the values $F(y_j)$:

$$Out_{i,j} = (F(y_j))(i) = \mathrm{ENC}_{pk_1}(0; r_{i,j}) \cdot \mathrm{ENC}_{pk_1}(P_{0,j}(i); 0) \cdot \mathrm{ENC}_{pk_1}(P_{y_j}(i); 0) \cdot \left(\prod_{s=0}^{n} (b_s)^{\Pi_{\ell=0}^{\lfloor \log n \rfloor} (P_{y_j^{2^\ell}}(i))^{s[\ell]}} \right)^{P_{r_j}(i)}$$

   and sends them to the Client.
9. The Client decrypts the values that he received from the Server, verifies that they are valid, and uses them to reconstruct the shared values. He concludes that the obtained values that are in his input set are the values in the intersection set.

The above protocol ensures privacy in the presence of semi-honest parties, but is not secure in the presence of malicious parties. The following are several basic additional conditions that must hold in order that the above protocol will be secure in the presence of malicious parties.

The first condition is that the coefficients that the Client sends to the Server are values encrypted with Hom_{enc} under the key pk. We guarantee this by making the Client prove that he knows the encrypted values with HEPKPV.

Additionally, the Client must be sure that the Server correctly shared his inputs using the secret-sharing scheme. This will be guaranteed by HEPKPV showing that all the shares of one input lie on some polynomial of degree k.

The correctness of the protocol also depends on the Server evaluating F honestly. We apply the cut-and-choose protocol on the shares of the Server's inputs to ensure that the computation on a large fraction of final output shares was done correctly.

The last change that we apply improves the efficiency of the protocol. Since the number of shares needed to reconstruct $F_j(0)$ will depend on the degree of F, we reduce its degree by introducing new variables of the form $a_i = y^{2^i}$ for $1 \leq i \leq \lfloor \log n - 1 \rfloor$ for $y \in Y$. Here we need to prove that the computation of the new variables and their shares was done correctly with the Preprocessing Verification Protocol.

We present the full set intersection protocol below.

Set Intersection Protocol Π

Input: $C \leftarrow \{X, max_c, max_s, Hom_{enc}, k\}, S \leftarrow \{Y, max_s, max_c, Hom_{enc}, k\}$
Output: $C \rightarrow X \cap Y, S \rightarrow \perp$
Protocol:

1. The Client runs GEN(1^k) to obtain a secret key $sk1$ and a public key $pk1$ for Hom_{enc} and sends $pk1$ to the Server.
2. C computes a polynomial $P(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ of degree $n = |X|$ such that $P(x_i) = 0$ if and only if $x_i \in X$. Let $a_n = 1$.
3. C computes $b_i = \text{ENC}_{pk1}(a_i)$ for all $0 \leq i \leq n-1$ and sends to S $\{b_{n-1}, \cdots, b_0\}$,
4. C and S run the HEPKPV Protocol presented in Section 2 as P_0 and P_1 respectively with common input: $\overline{B} = \{b_{n-1}, \cdots, b_0\}$ and $L = \{0,1\}^q$, in order that the C proves that it knows the decryptions of $\{b_{n-1}, \cdots, b_0\}$.
5. The Server runs GEN(1^k) to obtain a secret key $sk2$ and a public key $pk2$ for Hom_{enc} and sends $pk2$ to the Client.
6. For each $y_j \in Y$ S runs the Efficient Preprocessing protocol to obtain the new variables $c_\ell = y_j^{2^\ell}$ for $0 \leq \ell \leq \lfloor \log n - 1 \rfloor$ and the corresponding sharing polynomials $P_{y_j^{2^\ell}}$ such that $P_{y_j^{2^\ell}}(0) = y_j^{2^\ell}$. During the protocol S commits to $P_{y_j^{2^\ell}}(i)$ for $1 \leq j \leq |Y|, 1 \leq \ell \leq \lfloor \log n \rfloor + 1, 1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)$.
7. For each $y_j \in Y$ S chooses a random value r_j and selects a random polynomial P_{r_j} of degree k with constant coefficient equal to r_j, shares r_j into into $10k(\lfloor \log n \rfloor + 1)$ shares, and sends the following share commitments to C: $(\text{ENC}_{pk2}(P_{r_j}(1)), \ldots, \text{ENC}_{pk2}(P_{r_j}(10k(\lfloor \log n \rfloor + 1)))$
8. For each $y_j \in Y$ S chooses a random polynomial $P_{0,j}$ of degree $k + k(\lfloor \log n \rfloor + 1)$ with constant coefficient equal to 0, computes $10k(\lfloor \log n \rfloor + 1)$ shares, and sends the following share commitments to C: $(\text{ENC}_{pk2}(P_{0_j}(1)), \ldots, \text{ENC}_{pk2}(P_{0_j}(10k(\lfloor \log n \rfloor + 1)))$
9. For each $y_j \in Y$, for $1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)$ using the sharing polynomials obtained in Steps 6, 7, 8, and a random value $r_{i,j}$ S computes:

$$Out_{i,j} = \text{ENC}_{pk_1}(0; r_{i,j}) \cdot \text{ENC}_{pk_1}(P_{0,j}(i); 0) \cdot \text{ENC}_{pk_1}(P_{y_j}(i); 0) \cdot \left(\prod_{s=0}^{n} (b_s)^{\Pi_{\ell=0}^{\lfloor \log n \rfloor} (P_{y_j^{2^\ell}}(i))^{s[\ell]}} \right)^{P_{r_j}(i)}$$

where $s[\ell]$ denotes the ℓ^{th} bit of s and sends the obtained values to C.
10. C and S run the coin tossing protocol to choose a random number $R \in [1, |J|]$.
11. S and C run the Preprocessing Verification protocol with the share commitments that S computed in Step 6 in order for S to prove to C that it correctly computes the new variables and their shares.
12. S and C run the HEPKPV protocol as P_0 and P_1 respectively so that S proves to C knowledge and validity of the commitments $L_{poly}(10k(\lfloor \log n \rfloor + 1), |Y|, k) = \{m_{i,j} = P_{r_j}(i)\}$, $L_{poly,0}(10k(\lfloor \log n \rfloor + 1), |Y|, k + k(\lfloor \log n \rfloor + 1)) = \{m_{i,j} = P_{0,j}(i)\}$.
13. C and S run the cut-and-choose protocol to prove that S correctly computed $[Out_{i,j}]$.
14. C runs the Reconstruction Protocol to obtain the final output.

4 Analysis

Our main theorem is the following:

Theorem 1. *If the Decisional Diffie-Hellman problem is hard in G with generator q and protocol Π is instantiated with the additive El-Gamal encryption scheme such that $Hom_{enc} = AEG_{enc}$, then Π securely computes the Set Intersection functionality in the presence of malicious adversaries.*

We note that Π is also secure when instantiated with any homomorphic encryption scheme satisfying property 2, and allowing to solve the Lagrange interpolation, as discussed in Sections 2 and 3.3. In particular, we can securely instantiate the protocol with a properly modified version of Paillier encryption (but the details are left out of this abstract). The complete proof of Theorem 1 is in the full version of our paper. Here we give some intuition and a proof sketch.

4.1 Client-Side Simulator

We consider the case in which the Client is corrupted and the Server is honest.

Let A_C be a non-uniform probabilistic polynomial-time real adversary that controls the Client. We construct a non-uniform probabilistic expected polynomial-time ideal model adversary simulator S_C.

The idea behind how S_C works is that it first extracts the Malicious Client's inputs using the extractor for the HEPKPV protocol. S_C then plays the role of the Honest Server using dummy inputs that are all set to 0. When proving knowledge and validity of the Server's input, S_S uses the simulator for the HEPKPV protocol. Next, the Simulator chooses a random subset I' of size k such that $I' \subset [10k(\lfloor \log n \rfloor + 1)]$. When committing to the secret-sharing of its input, it places random values in the positions indexed by I'. S_C computes correctly all calculations that will be verified in the cut-and-choose step for elements in the subset I'. Then, S_C simulates the Coin-Tossing protocol to guarantee that the outcome of the Coin-Tossing protocol is $I = I'$. To ensure that the final output sent to the Client is correct, the Simulator utilizes the the Trusted Party to find out the elements in $X \cap Y$ and includes them in the Server's final output. Intuitively, because the Simulator is able to choose the set I ahead of time, the Simulator can run the protocol using the challenge ciphertext from a CPA-IND experiment as the inputs of the Server in indeces $i \notin I$, thereby reducing indistinguishability of the views to the semantic security of the encryption scheme AEG_{enc}. Therefore, we have that the Malicious Client cannot distinguish its view in the Ideal Model when interacting with a Simulator that chooses all 0 values as the Server's input for indeces $i \notin I$ and its view in the Real model when the Honest Server uses its actual input. This is due to the information-theoretic secrecy of the secret-sharing scheme and the semantic security of the encryption scheme.

We now describe in detail the Simulator for the case of the Malicious Client and Honest Server

1. S_C extracts the Client's inputs using the extractor for the HEPKPV protocol.
2. S_C uses the Berlekamp factoring algorithm ([2]) to factor the extracted polynomial and obtain the Malicious Client's input set X.
3. S_C sends X to the Trusted Party and receives back the set $Out = X \cap Y$.
4. S_C chooses a random subset $I' \subset [10k(\lfloor logn \rfloor + 1)]$ of size k, $I' = \{j_1, \ldots, j_k\}$
5. **Input Preprocessing:**
   - S_C chooses a random value $r_{i,j,l}$ and sets $P_{y_j^{2^i}}(l) = r_{i,j,l}$ for $1 \leq j \leq max_S, 0 \leq i \leq \lfloor \log n \rfloor, l \in I$.
   - S_C sets $P_{y_j^{2^i}}(l) = 0$ for $1 \leq j \leq max_S, 0 \leq i \leq \lfloor \log n \rfloor, l \in [10k(\lfloor logn \rfloor + 1)] \setminus I'$.
   - S_C sets $P^2_{y_j^{2^i}}(l) = (P_{y_j^{2^i}}(l))^2$ for $1 \leq j \leq max_S, 0 \leq i \leq \lfloor \log n \rfloor - 1, l \in I'$
   - S_C sets $P^2_{y_j^{2^i}}(l) = 0$ for $1 \leq j \leq max_S, 0 \leq i \leq \lfloor \log n \rfloor - 1, l \in [10k(\lfloor logn \rfloor + 1)] \setminus I'$
   - S_C commits to these inputs.
6. **Choosing Random Polynomials:**
   - S_C chooses a random value $r_{j,l}$ and sets $P_{r_j}(l) = r_{j,l}$ for $1 \leq j \leq max_S, l \in I'$.
   - S_C sets $P_{r_j}(l) = 0$ for $1 \leq j \leq max_S, l \in [10k(\lfloor logn \rfloor + 1)] \setminus I'$.
   - S_C commits to these inputs
7. **Choosing Zero Polynomials:**
   - S_C chooses a random value $r_{j,l}$ and sets $P_{0,j} = r_{j,l}$ for $1 \leq j \leq max_S, l \in I'$.
   - S_C sets $P_{0,j} = 0$ for $1 \leq j \leq max_S, l \in [10k(\lfloor logn \rfloor + 1)] \setminus I'$.
   - S_C commits to these inputs
8. For $1 \leq j \leq max_S$ and for $i \in I'$, S_C honestly computes the outputs $Out_{i,j} = \text{ENC}_{pk1}(s_{i,j})$ based on the inputs committed to in the previous stages.
9. For each $y_j \in Out$, S_C chooses a random polynomial P_{Out_j} of degree $k + k(\lfloor \log n \rfloor + 1)$ such that $P_{Out_j}(i) = s_{i,j}$ for $i \in I'$ and $P_{Out_j}(0) = y_j$. Note that S_C can compute $s_{i,j}$ since it has extracted the coefficients of the Client's polynomial P.
10. For each $y_j \in V$, S_C chooses a random polynomial P_{Out_j} such that $P_{Out_j}(i) = s_{i,j}$ for $i \in I'$.
11. For $Out_{i,j}$, $i \in [10k(\lfloor logn \rfloor + 1)] \setminus I', 1 \leq j \leq max_S$ S_C computes a random encryption of $P_{Out_j}(i)$.
12. S_C commits to the shares of its inputs and sends the output computed above to A_C.
13. S_C simulates a run of the HEPKPV protocol with the committed inputs from above using the simulator for the HEPKPV protocol.
14. S_C simulates a run of the Coin-Tossing protocol to ensure the outcome is the set $I' = J_R$ using the simulator for the Coin-Tossing protocol.
15. S_C plays the role of the honest Server in the Preprocessing Verification protocol to prove the preprocessing was done correctly.
16. S_C plays the role of the honest Server in the the Cut-and-Choose protocol to prove output was calculated correctly.

4.2 Sender-Side Simulator

We now consider the case in which the Sender is corrupted and the Receiver is honest.

Let A_S be a non-uniform probabilistic polynomial-time real adversary that controls the Server. We construct a non-uniform probabilistic expected polynomial-time ideal model adversary simulator S_S.

The idea behind how S_S works is that it plays the role of the Honest Client using dummy inputs: For the coefficients of the polynomial P, it sends n random encryptions of 0. Then, instead of playing the role of the prover in the zero knowledge proof of knowledge for the validity and knowledge of the coefficients of P, it invokes the Simulator for the HEPKPV protocol. In the second stage, S_S uses the extractability property of the HEPKPV to obtain the inputs of the Malicious Server from the Share Commitment Protocol. After obtaining all the input commitments and output from the Server, S_S continues to play the role of the Honest Client in the coin-tossing protocol to choose a random subset for the cut-and-choose test. If the Malicious Server passes the cut-and-choose test, then the inputs extracted earlier are submitted to the Trusted Party, otherwise the Simulator aborts (as the Honest Client does). The cut-and-choose test ensures that most of the shares of the output generated by the Malicious Server are consistent with the input extracted previously. Additionally, the honest Client in the Real Model checks that he has, in fact, received a polynomial. Due to the properties of the secret-sharing scheme, the above two points ensure that with all but negligible probability the same (correct) output will be obtained by an Honest Client in the Real and Ideal model.

We now describe in detail the Simulator for the case of the Malicious Server and Honest Client

1. S_S chooses n random encryptions of 0: $c_1, \ldots c_n$ and sends to Server.
2. S_S simulates a run of the HEPKPV protocol with input from above using the simulator for the HEPKPV protocol.
3. S_S extracts the Server's inputs using the extractor for the HEPKPV protocol.
4. A_S computes output $v_{i,j}$ for $i = 1$ to $10k(\lfloor \log n \rfloor + 1)$ and $j = 1$ to $|Y|$ and sends to Simulator
5. S_S plays the part of the Honest Client in the Coin-Tossing protocol.
6. S_S and A_S run the Cut-and-Choose protocol.
7. S_S rewinds A_S to the beginning of the Coin-Tossing protocol and re-runs the protocol with new randomness
8. S_S and A_S run the Cut-and-Choose protocol.
9. S_S repeats the previous two steps until all input indeces from 1 to $10k(\lfloor \log n \rfloor + 1)$ have been opened and the output $v_{i,j}$ has been shown to be computed correctly.
10. S_S submits the previously extracted inputs to the TP.

4.3 Computation and Communication Complexity

The communication complexity of our protocol is $O(mk^2 \log^2 n + kn)$ encryptions and the computational complexity is $O(mnk \log n + mk^2 \log^2 n)$ exponentiations.

The best known protocols for Set Intersection secure against malicious parties until now were generic protocols based on Yao's garbled circuit ([24,25]). Clearly, the communication complexity of these protocols must at least be the size of the circuit required for the functionality since all generic two-party protocols that are known require one party to send a (garbled) circuit for the functionality being evaluated.

The best known circuit for evaluating the Set Intersection functionality has size $O(m \cdot n)$, where m and n are the size of the Server and Client's inputs respectively, since we must have at least $O(m \cdot n)$ comparisons to compute the functionality. A secure implementation will require a bit-wise circuit of size at least $O(m \cdot n \cdot k)$, and this does not even take into account the costly zero-knowledge techniques that must be employed. Our communication complexity of $O(mk^2 \log^2 n + kn)$ is much smaller.

Additionally, our protocol accesses the underlying field in a black-box manner. This is in contrast to an implementation based on a Yao circuit (which must be binary) that is used in the generic protocols for 2-party computation. Therefore, our complexity scales much better as the size of the field increases.

Acknowledgment. We thank Stas Jarecki for very helpful discussions.

References

1. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: SIGMOD 2003: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 86–97. ACM, New York (2003)
2. Berlekamp, E.: Factoring polynomials over large finite fields. Mathematics of Computation 24, 713–735 (1970)
3. Boudot, F., Schoenmakers, B., Traoré, J.: A fair and efficient solution to the socialist millionaires problem. Discrete Applied Mathematics 111, 2001 (2001)
4. Camenisch, J., Zaverucha, G.: Private intersection of certified sets. In: Proceedings of Financial Cryptography 2009 (2009)
5. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13, 2000 (2000)
6. Choi, S., Dachman-Soled, D., Malkin, T., Wee, H.: Black-box construction of a non-malleable encryption scheme from any semantically secure one. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 427–444. Springer, Heidelberg (2008)
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)
8. Fagin, R., Naor, M., Winkler, P.: Comparing information without leaking it. Communications of the ACM 39, 77–85 (1996)
9. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)

10. Goldreich, O.: Foundations of cryptography: a primer. Found. Trends Theor. Comput. Sci. 1(1), 1–116 (2005)
11. Shafi, G., Silvio, M.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In: STOC 1982: Proceedings of the fourteenth annual ACM symposium on Theory of computing, pp. 365–377. ACM, New York (1982)
12. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (2008)
13. Impagliazzo, R., Yung, M.: Direct minimum knowledge computations. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 40–51. Springer, Heidelberg (1988)
14. Jakobsson, M., Yung, M.: Proving without knowing: On oblivious, agnostic and blindfolded provers. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 186–200. Springer, Heidelberg (1996)
15. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: TCC, pp. 577–594 (2009)
16. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman & Hall/Crc Cryptography and Network Security Series. Chapman & Hall/CRC, Boca Raton (2007)
17. Kiayias, A., Mitrofanova, A.: Testing disjointness of private datasets. In: Patrick, A.S., Yung, M. (eds.) FC 2005. LNCS, vol. 3570, pp. 109–124. Springer, Heidelberg (2005)
18. Kissner, L., Song, D.X.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
19. Lindell, Y., Pinkas, B.: Privacy preserving data mining. Journal of Cryptology, 36–54 (2000)
20. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
21. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: STOC 1999: Proceedings of the thirty-first annual ACM symposium on Theory of computing, pp. 245–254. ACM Press, New York (1999)
22. 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)
23. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
24. Yao, A.C.-C.: Protocols for secure computations. In: FOCS, pp. 160–164 (1982)
25. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: FOCS, pp. 162–167 (1986)

A New Variant of the Cramer-Shoup KEM Secure against Chosen Ciphertext Attack

Joonsang Baek[1], Willy Susilo[2], Joseph K. Liu[1], and Jianying Zhou[1]

[1] Cryptography and Security Department
Institute for Infocomm Research, Singapore
{jsbaek,ksliu,jyzhou}@i2r.a-star.edu.sg

[2] Centre for Computer and Information Security Research
School of Computer Science and Software Engineering
University of Wollongong, Australia
wsusilo@uow.edu.au

Abstract. We propose a new variant of the Cramer-Shoup KEM (key encapsulation mechanism). The proposed variant is more efficient than the original Cramer-Shoup KEM scheme in terms of public key size and encapsulation cost, but is proven to be (still) secure against chosen ciphertext attack in the standard model, relative to the Decisional Diffie-Hellman problem.

1 Introduction

Motivation. At Crypto '98, Cramer and Shoup [9] proposed the first practical public key encryption (PKE) scheme whose security against adaptive chosen ciphertext attack (which we simply call "CCA" throughout this paper) can be proven without depending on the random oracle model [6]. This is a striking result as the chosen ciphertext security without random oracles could be achieved by only adding a few more exponentiations to the original ElGamal encryption scheme, in contrast to the computationally heavy solutions [11,20] based on zero-knowledge proofs proposed before. Nearly seven years later, a major improvement on the performance of the Cramer-Shoup PKE scheme was made by Kurosawa and Desmedt [17]. They were able to obtain a very efficient hybrid PKE scheme by simplifying the Cramer-Shoup PKE scheme with the help of the "ciphertext authenticity checking" mechanism of the underlying symmetric encryption primitive. Afterwards, Hofheinz and Kiltz [14] came up with a dual version of the Kurosawa-Desmedt PKE scheme. Note that chosen ciphertext security of all these schemes are relative to the (standard) Decisional Diffie-Hellman (DDH) problem.

In the full version of their Crypto '98 paper, Cramer and Shoup [10] formulated a framework called "KEM/DEM (Key Encapsulation Mechanism/Data Encapsulation Mechanism)". A KEM is a public key scheme that outputs a (session) key taking public key as input. According to the KEM/DEM framework, a (hybrid) PKE scheme secure against CCA can be constructed in such a way that a key output by

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 143–155, 2009.

a CCA-secure KEM scheme[1] is used as a session key for an one-time CCA-secure DEM (i.e., symmetric encryption) scheme that encrypts a plaintext message.

In the same paper, Cramer and Shoup proposed a KEM scheme based on their original PKE scheme, which we denote by "CS-KEM", and showed it is CCA-secure assuming that the DDH problem is hard. Interestingly, however, it was shown [13] that the KEM scheme extracted from Kurosawa and Desmedt's hybrid PKE scheme, which we denote by "KD-KEM", does not satisfy full CCA-security even though the hybrid PKE scheme remains secure against CCA. Abe et al. [1] showed later that the KD-KEM scheme is actually secure against "LCCA (predicate-dependent CCA)" which is weaker than usual CCA-security of KEM. Similarly, the KEM scheme extracted from Hofheinz and Kiltz's [14] hybrid PKE scheme, denoted "HK-KEM", was shown to be secure against CCCA (constrained CCA), which is also weaker than the usual CCA-security of KEM.

Hence, the CS-KEM scheme is, though less efficient than the KD-KEM and HK-KEM schemes, the only KEM scheme that is fully CCA-secure without random oracles, assuming that the DDH problem is hard. A remaining question is whether the performance of the CS-KEM scheme can be further improved. In this paper, we give a positive answer to this question.

Recent Developments. In 2007, Kiltz [16] proposed a KEM scheme whose CCA security is based on the gap hashed Diffie-Hellman problem. An interesting feature of this scheme is that different from the CS-KEM scheme, a key can be computed from one of the public key components used to create one ciphertext component. More precisely, let $pk = (q, g, c, d)$ be public key, where g is a generator of a group of prime order q; $c = g^x$ and $d = g^y$ for some random $(x, y) \in \mathbb{Z}_q^*$. In this scheme, a ciphertext and its corresponding key is computed as $(g^r, (c^\alpha d)^r)$ and $\mathsf{KDF}(c^r)$ respectively, where KDF denotes a key derivation function. As mentioned earlier, the public c used to create $(c^\alpha d)^r$ is reused to produce c^r. Note here that one cannot expect a computational gain even if c is reused. However, if d were reused, a computational cost could be reduced by computing $c^{r\alpha}$ and d^r separately to generate $(c^\alpha d)^r$ and using d^r as a key. Indeed, Lu et al. [18] recently showed that this modified version of Kiltz's KEM scheme is CCA-secure.

More recently, as one of the applications of their new computational problem called "Twin Diffie-Hellman", Cash et al. [8] proposed a new variant of Cramer and Shoup's PKE scheme and showed that it is CCA-secure under the hashed decisional Diffie-Hellman assumption, which is weaker than the usual DDH assumption[2]. Although this variant has interesting theoretical implications, it is computationally more expensive than the original Cramer and Shoup's one and hence ours.

Our Contributions. We observe that it is also possible to apply the structure of Kiltz's KEM scheme to the CS-KEM scheme. As a result, we could construct a KEM scheme which is proven to be *fully* CCA-secure without random oracles assuming that the DDH problem is hard, while it is more efficient than

[1] The CCA security notion for KEM will be defined in Section 2.

[2] Note that although [8] focuses only on a PKE scheme, a corresponding KEM scheme can easily be derived and analyzed in an obvious way.

the CS-KEM scheme. The crux is the efficiency of our scheme in terms of a shorter public/private key pair and improved encapsulation speed. However, we honestly state that the improvement on the encapsulation speed would not be very much dramatic due to the advancement of fast multi-exponentiation algorithms [2,7,19], which makes the cost for computing double exponentiation very close to the cost of computing a single exponentiation. Nevertheless, the proposed scheme has a new structure, which reduces one group element of the public key of the CS-KEM scheme. We believe it is also theoretically interesting in that it shows yet another way of constructing a more efficient variant of the CS-KEM without sacrificing full CCA-security.

2 Preliminaries

In this section, we review the formal notion of key encapsulation mechanism (KEM) and its security against adaptive chosen ciphertext attack (CCA). We also review building blocks used in our construction of KEM which will be presented in Section 3.

Key Encapsulation Mechanism (KEM). The KEM scheme, denoted KEM, consists of the following algorithms [10,15,22].

- Key Generation: Taking 1^λ for a security parameter $\lambda \in \mathbb{Z}_{\geq 0}$ as input, this algorithm generates a public/private key pair (pk, sk).
- Encapsulation: Taking 1^λ and a public key pk as input, this algorithm generates a ciphertext/(symmetric) key pair (ψ, K).
- Decapsulation: Taking 1^λ, a private key sk and a ciphertext ψ as input, this algorithm outputs either a (symmetric) key K or the special symbol $\perp$, meaning "reject".

The security against CCA of KEM is defined as follows. Consider any attacker $\mathcal{A}$ and any value $\lambda > 0$ for security parameter in the following game $\mathsf{GameCCA}_{\mathcal{A}}^{\mathsf{KEM}}(\lambda)$ in which $\mathcal{A}$ interacts with the challenger.

Phase 1: The challenger runs the key generation algorithm providing 1^λ as input to generate a public/private key pair (pk, sk). The challenger then computes a challenge ciphertext ϕ^* and a key K_1^* by running the encapsulation algorithm. It also picks $K_0^* \in S_K$ at random, where S_K denotes the key space. It then picks $\beta \in \{0, 1\}$ at random and gives (pk, ϕ^*, K_β^*) to $\mathcal{A}$.
Phase 2: $\mathcal{A}$ submits ciphertexts, each of which is denoted by ϕ. On receiving ϕ, the challenger runs the decapsulation algorithm on input ϕ and passes the resulting decapsulation to $\mathcal{A}$. At the end of this phase, $\mathcal{A}$ outputs its guess $\beta' \in \{0, 1\}$.
We define the output of the game to be 1 if $\beta' = \beta$, and 0 otherwise. $\mathcal{A}$'s success is defined by the probability

$$\mathbf{Adv}_{\mathcal{A},\mathsf{KEM}}^{\mathrm{CCA}}(\lambda) = \left| \Pr[\mathsf{GameCCA}_{\mathcal{A}}^{\mathsf{KEM}}(\lambda) = 1] - \frac{1}{2} \right|.$$

We say that KEM is CCA-secure if $\mathbf{Adv}_{\mathsf{KEM}}^{\mathrm{CCA}}(\lambda) = \max_{\mathcal{A}} \left\{ \mathbf{Adv}_{\mathcal{A},\mathsf{KEM}}^{\mathrm{CCA}}(\lambda) \right\}$ is negligible for any attacker $\mathcal{A}$.

The Decisional Diffie-Hellman Problem. We now review the definition of the Decisional Diffie-Hellman (DDH) problem. Let $\mathcal{D}$ be an attacker. Let $\mathbb{G}$ be a finite cyclic group generated by $g \in \mathbb{G}$. Let q be a prime order of $\mathbb{G}$, whose size depends on the security parameter λ. We define the DDH problem using the attacker $\mathcal{D}$'s advantage in distinguishing two distributions:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{DDH}}_{\mathcal{D},\mathbb{G}}(\lambda) = |\Pr[a \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q; b \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q : 1 \leftarrow \mathcal{D}(1^\lambda, g^a, g^b, g^{ab})]\\ - \Pr[a \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q; b \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q; r \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q : 1 \leftarrow \mathcal{D}(1^\lambda, g^a, g^b, g^r)]|.\end{aligned}$$

Equivalently [9,10,12],

$$\begin{aligned}\mathbf{Adv}^{\mathrm{DDH}}_{\mathcal{D},\mathbb{G}}(\lambda) = |\Pr[w \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q; g_2 \leftarrow g_1^w; r \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q : 1 \leftarrow \mathcal{D}(1^\lambda, g_1, g_2, g_1^r, g_2^r)]\\ - \Pr[w \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q; g_2 \leftarrow g_1^w; r' \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q \setminus \{r\} : 1 \leftarrow \mathcal{D}(1^\lambda, g_1, g_2, g_1^r, g_2^{r'})]|,\end{aligned}$$

where g_1 is the generator of $\mathbb{G}$.

We say that the DDH problem is hard if $\mathbf{Adv}^{\mathrm{DDH}}_{\mathbb{G}}(\lambda) = \max_{\mathcal{D}} \left\{\mathbf{Adv}^{\mathrm{CCA}}_{\mathcal{D},\mathbb{G}}(\lambda)\right\}$ is negligible for any attacker $\mathcal{D}$.

Target Collision Resistant Hash Function (TCR). The security of the target collision resistant hash function denoted by H is defined as follows. Given a n tuple of group elements $x \in \mathbb{G}^n$, it is hard for an attacker $\mathcal{B}_1$ to find $y \neq x$ such that $\mathsf{H}(x) = \mathsf{H}(y)$. We define the attacker $\mathcal{B}_1$'s success probability by $\mathbf{Adv}^{\mathrm{COL}}_{\mathcal{B}_1,\mathsf{H}}(\lambda)$. We say that H is target collision-resistant if $\mathbf{Adv}^{\mathrm{COL}}_{\mathsf{H}}(\lambda) = \max_{\mathcal{B}_1} \left\{\mathbf{Adv}^{\mathrm{COL}}_{\mathcal{B}_1,\mathsf{H}}(\lambda)\right\}$ is negligible for any attacker $\mathcal{B}_1$.

Key Derivation Function (KDF). In the proposed variant of the KD-KEM scheme, we will use the key derivation function denoted by KDF. Specifically, KDF takes two random elements a and b in the group $\mathbb{G}$ as input. Let l be the length of the output of KDF, which depends on the security parameter λ. We define the security of KDF with respect to an attacker $\mathcal{B}_2$ as follows. (Below, "ROR" stands for "real or random".)

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ROR}}_{\mathcal{B}_2,\mathsf{KDF}}(\lambda) = |\Pr[a, b \overset{\mathrm{R}}{\leftarrow} \mathbb{G} : 1 \leftarrow \mathcal{B}_2(1^\lambda, a, \mathsf{KDF}(a, b))]\\ - \Pr[a \overset{\mathrm{R}}{\leftarrow} \mathbb{G}; \mu \overset{\mathrm{R}}{\leftarrow} \{0,1\}^l : 1 \leftarrow \mathcal{B}_2(1^\lambda, a, \mu)]|.\end{aligned}$$

We say that KDF is secure if $\mathbf{Adv}^{\mathrm{ROR}}_{\mathsf{KDF}}(\lambda) = \max_{\mathcal{B}_2} \left\{\mathbf{Adv}^{\mathrm{ROR}}_{\mathcal{B}_2,\mathsf{KDF}}(\lambda)\right\}$ is negligible for any attacker $\mathcal{B}_2$.

Notice that the above security requirement on KDF is the same as that of the KDF functions used in [10].

3 The Proposed Variant of the Cramer-Shoup KEM

Description. We describe our variant of the CS-KEM scheme, which we denote by "Π", as follows. (Readers are referred to the end of Section 1 for the underlying idea of our scheme.)

Key Generation: Pick a group $\mathbb{G}$ of prime order q and generators g_1 and g_2 of $\mathbb{G}$. Pick a target-collision resistant hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q^*$ and a key derivation function KDF. Then choose $(x_1, x_2, y_1, y_2) \in \mathbb{Z}_q^4$ at random and compute

$$c = g_1^{x_1} g_2^{x_2},\ d = g_1^{y_1} g_2^{y_2}.$$

Return public key $pk = (\mathbb{G}, q, g_1, g_2, c, d, \mathsf{H}, \mathsf{KDF})$ and private key $sk = (pk, x_1, x_2, y_1, y_2)$.
Encapsulation: Pick $r \in \mathbb{Z}_q^*$ at random and compute

$$u_1 = g_1^r,\ u_2 = g_2^r,\ \alpha = \mathsf{H}(u_1, u_2),\ v = c^r d^{r\alpha},\ K = \mathsf{KDF}(u_1, c^r).$$

Return a ciphertext $\psi = (u_1, u_2, v)$ and a key K.
Decapsulation: Upon receiving $\psi = (u_1, u_2, v)$, compute

$$\alpha = \mathsf{H}(u_1, u_2),\ v' = u_1^{x_1 + y_1\alpha} u_2^{x_2 + y_2\alpha},\ K = \mathsf{KDF}(u_1, u_1^{x_1} u_2^{x_2})$$

If $v' = v$ then return K; otherwise, return $\perp$.

We show that the scheme Π is CCA-secure, relative to the DDH problem. More precisely, we prove the following theorem.

Theorem 1. *The KEM scheme Π is CCA-secure assuming that the DDH problem is hard and the underlying hash function* H *s target collision-resistant and key derivation function* KDF *is secure. More precisely, we have*

$$\mathbf{Adv}_{\Pi}^{CCA}(\lambda) \leq \mathbf{Adv}_{\mathbb{G}}^{DDH}(\lambda) + \mathbf{Adv}_{\mathsf{H}}^{COL}(\lambda) + \mathbf{Adv}_{\mathsf{KDF}}^{ROR}(\lambda) + \frac{q_D}{q}.$$

where λ denotes the security parameter and q_D is the number of queries to the decapsulation oracle.

Outline of Proof. The basic idea of the proof essentially follows the logic of the proofs of the CS-KEM [10] and CS-PKE [9] schemes. We need to show that by using a CCA-attacker for the scheme Π as a subroutine, a DDH attacker can solve the DDH problem: When the DDH attacker is given a right Diffie-Hellman tuple (g_1, g_2, g_1^r, g_2^r), it can perfectly simulate the environment of the CCA-attacker. On the other hand, when it is given $(g_1, g_2, g_1^r, g_2^{r'})$ where $r' \neq r$, the output of the decapsulation oracle will not be legitimate but we will show that this one won't be a problem.

In our proof, there is an important difference from the proofs of the CS-KEM/CS-PKE schemes. Since the public key component c used to create $v = c^r d^{r\alpha}$ is "reused" to produce a key material c^r, we need to assume that the attacker's view include c, d, v and c^r when breaking the confidentiality (i.e. "key indistinguishability") of the scheme Π. (Note that this is different from the CS-KEM/CS-PKE schemes in which an independent public key component is used to produce a key.) By using an argument from linear algebra, we show that fortunately, this does not cause a problem. (In particular, readers are referred to Equation (12)).

Proof. Fix an attacker $\mathcal{A}$ that breaks CCA-security of the scheme Π. Also, fix an attacker $\mathcal{D}$ that is to solve the DDH problem.

Simulation. The DDH attacker $\mathcal{D}$ simulates the environment of $\mathcal{A}$ as follows. Assume that $\mathcal{D}$ is given a DDH instance (g_1, g_2, u_1, u_2) where g_1 and g_2 are generators of a group $\mathbb{G}$ of prime-order q. $\mathcal{D}$ chooses $(x_1, x_2, y_1, y_2) \in \mathbb{Z}_q^4$ at random and computes $c = g_1^{x_1} g_2^{x_2}$ and $d = g_1^{y_1} g_2^{y_2}$. $\mathcal{D}$ also chooses a hash function H and a key derivation function KDF, and gives $pk = (\mathbb{G}, q, g_1, g_2, c, d, \mathsf{H}, \mathsf{KDF})$ as a public key to $\mathcal{A}$.

When $\mathcal{A}$ queries ciphertexts to the decapsulation oracle in the find stage, $\mathcal{D}$ decapsulates them using (x_1, x_2, y_1, y_2).

$\mathcal{D}$ simulates the challenge ciphertext and the key as follows. $\mathcal{D}$ first sets $u_1^* = u_1$ and $u_2^* = u_2$, and computes $\alpha^* = \mathsf{H}(u_1^*, u_2^*)$, $v^* = (u_1^*)^{x_1 + y_1\alpha^*}(u_2^*)^{x_2 + y_2\alpha^*}$ and $K_1^* = \mathsf{KDF}(u_1^*, (u_1^*)^{x_1}(u_2^*)^{x_2})$. $\mathcal{D}$ also chooses K_0^* at random from the output space of KDF and picks $\beta \in \{0, 1\}$ at random. $\mathcal{D}$ finally gives $\mathcal{A}$ the challenge ciphertext-key pair (ψ^*, K_β^*) where $\psi^* = (u_1^*, u_2^*, v^*)$.

When $\mathcal{A}$ queries ciphertexts, all of which are different from ψ^*, to the decapsulation oracle in the find stage, $\mathcal{D}$ decapsulates them using (x_1, x_2, y_1, y_2).

Finally, when $\mathcal{A}$ outputs its guess β', $\mathcal{D}$ outputs 1 if $\beta' = \beta$; otherwise, outputs 0.

Analysis. We first analyze the case when $\mathcal{D}$ is given $(g_1, g_2, g_1^{r^*}, g_2^{r^*})$. First, we prove the following lemma.

Lemma 1

$$\Pr[\mathcal{D}(1^\lambda, g_1, g_2, g_1^{r^*}, g_2^{r^*}) = 1] = \Pr[\mathsf{GameCCA}_{\mathcal{A}}^{\Pi}(\lambda) = 1]. \tag{1}$$

Proof. Note that since (x_1, x_2, y_1, y_2) is randomly chosen from $\mathbb{Z}_q^4$, the public key pk is distributed the same as the public key in the real attack.

By the simulation of the challenge ciphertext presented above, we have

$$\psi^* = (u_1^*, u_2^*, v^*) = (g_1^{r^*}, g_2^{r^*}, (g_1^{r^*})^{x_1 + y_1\alpha^*}(g_2^{r^*})^{x_2 + y_2\alpha^*}) = (g_1^{r^*}, g_2^{r^*}, c^{r^*} d^{r^*\alpha^*})$$

and

$$K_1^* = \mathsf{KDF}(u_1^*, (g_1^{r^*})^{x_1}(g_2^{r^*})^{x_2}) = \mathsf{KDF}(u_1^*, c^{r^*}).$$

Since K_0^* is drawn uniformly at random from the output space of KDF, (ψ^*, K_β^*) has the right distribution.

It remains to show that the output of the decapsulation oracle (both in the simulation and the real attack) has the right distribution. Now, we call a ciphertext $\psi = (u_1, u_2, v)$ is invalid if $\log_{g_1} u_1 \neq \log_{g_2} u_2$. We show that invalid ciphertexts are rejected except for negligible probability.

First, by the public key pk that $\mathcal{A}$ sees, we have the following equations:

$$\log_{g_1} c = x_1 + x_2 w \tag{2}$$

and

$$\log_{g_1} d = y_1 + y_2 w, \tag{3}$$

where $w = \log_{g_1} g_2$. Hence, one can view (x_1, x_2, y_1, y_2) as a random point on the plane defined by (2) and (3). From the challenge ciphertext, we have

$$\log_{g_1} v^* = r^*(x_1 + x_2 w + y_1 \alpha^* + y_2 w \alpha^*), \tag{4}$$

where $r^* = \log_{g_1} u_1^* = \log_{g_2} u_2^*$ and $\alpha^* = \mathsf{H}(u_1^*, u_2^*)$. Note that the challenge ciphertext (whether it is in the simulation or real attack) does not constrain (x_1, x_2, y_1, y_2) as the hyperplane defined by (4) contains the plane formed by the equations (2) and (3). Now consider the following equation obtained from the invalid ciphertext ψ:

$$\log_{g_1} v = r_1 x_1 + r_2 x_2 w + r_1 y_1 \alpha + r_2 y_2 w \alpha, \tag{5}$$

where $r_1 = \log_{g_1} u_1$ and $r_2 = \log_{g_1} u_2$ such that $r_1 \neq r_2$. If the decapsulation oracle does not reject ψ, the point (x_1, x_2, y_1, y_2) should lie on the hyperplane defined by (5). But observe that the equations (2), (3) and (5) are linearly independent, so the hyperplane defined by (5) intersects the plane formed by the equations (2) and (3) at a line. This happens with probability $1/q$, which is negligible.

We now analyze the case when $\mathcal{D}$ is given $(g_1, g_2, g_1^{r_1^*}, g_2^{r_2^*})$ where $r_1^* \neq r_2^*$. More precisely, we prove the following lemma.

Lemma 2

$$\Pr[\mathcal{D}(1^\lambda, g_1, g_2, g_1^{r_1^*}, g_2^{r_2^*}) = 1] \leq \frac{1}{2} + \mathbf{Adv}_{\mathsf{KDF}}^{ROR}(\lambda) + \mathbf{Adv}_{\mathsf{H}}^{COL}(\lambda) + \frac{q_D}{q}. \tag{6}$$

The above bound (6) can be obtained by proving the following claims (1) and (2).

Recall that if a ciphertext $\psi = (u_1, u_2, v)$ is "invalid" then $\log_{g_1} u_1 \neq \log_{g_2} u_2$. We first prove the following claim.

Claim (1). Let $\mathsf{RejInvC}$ to be an event that the decapsulation oracle rejects all invalid ciphertexts. Then we have

$$\Pr[\beta' = \beta | \mathsf{RejInvC}] \leq \frac{1}{2} + \mathbf{Adv}_{\mathsf{KDF}}^{\mathrm{ROR}}(\lambda). \tag{7}$$

Proof of Claim (1). First, assume that that the decapsulation oracle rejects all invalid ciphertexts. We consider the distribution of the point $(x_1, x_2, y_1, y_2) \in \mathbb{Z}_q^4$ conditioned on $\mathcal{A}$'s view. Since the decapsulation oracle decapsulates only valid ciphertexts by the assumption (the decapsulation oracle rejects all invalid ciphertexts), for each ciphertext (u_1, u_2, v), $\mathcal{A}$ gets only linearly dependent relations

$$\log_{g_1} v = r(x_1 + x_2 w + y_1 \alpha + y_2 w \alpha), \tag{8}$$

and

$$\log_{g_1} u_1^{x_1} u_2^{x_2} = r(x_1 + x_2 w), \tag{9}$$

where $r = \log_{g_1} u_1 = \log_{g_2} u_2$ and $\alpha = \mathsf{H}(u_1, u_2)$. (In fact, $\mathcal{A}$ only gets the key which is the output of KDF which "wraps" the key material $u_1^{x_1} u_2^{x_2}$.) Hence, no information about the point (x_1, x_2, y_1, y_2) is leaked from querying valid ciphertexts to the decapsulation oracle.

Now consider the challenge ciphertext $\psi^* = (u_1^*, u_2^*, v^*)$ and the key $K_1^* = \mathsf{KDF}(u_1^*, (u_1^*)^{x_1}(u_2^*)^{x_2})$, produced by the simulation. Suppose that $\mathcal{A}$ gets the key material $(u_1^*)^{x_1}(u_2^*)^{x_2}$ at the worst case. Since v^* and $(u_1^*)^{x_1}(u_2^*)^{x_2}$ are in $\mathcal{A}$'s view, (x_1, x_2, y_1, y_2) should then satisfy the following equations

$$\log_{g_1} v^* = r_1^* x_1 + r_2^* x_2 w + r_1^* y_1 \alpha^* + r_2^* y_2 w \alpha^*, \tag{10}$$

where $r_1^* = \log_{g_1} u_1^*$, $r_2^* = \log_{g_2} u_2^*$ with $r_1^* \neq r_2^*$ and $\alpha^* = \mathsf{H}(u_1^*, u_2^*)$, and

$$\log_{g_1} (u_1^*)^{x_1}(u_2^*)^{x_2} = r_1^* x_1 + r_2^* x_2 w. \tag{11}$$

Now observe that

$$\det \begin{bmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ r_1^* & r_2^* w & r_1^* \alpha^* & r_2^* \alpha^* w \\ r_1^* & r_2^* w & 0 & 0 \end{bmatrix} = w^2 \alpha^* (r_1^* - r_2^*)^2 \neq 0. \tag{12}$$

Hence, the equations (2), (3), (10) and (11) are linearly independent. Note that $(u_1^*)^{x_1}(u_2^*)^{x_2}$ is distributed uniformly in $\mathbb{G}$ since r_1^* and r_2^* are chosen uniformly at random from $\mathbb{Z}_q$ and that K_0^* has been chosen uniformly at random and independently from anything else. Thus the distribution of β is independent from $\mathcal{A}$'s view under the assumption that KDF is secure and we get the bound (7). This is the end of proof of *Claim (1).*

We now show that the probability that the decapsulation oracle does *not* reject all invalid ciphertexts, i.e. $\Pr[\neg\mathsf{RejInvC}]$, is bounded by insecurity of hash function and some negligible probability. Precisely we prove the following claim.

Claim (2).

$$\Pr[\neg\mathsf{RejInvC}] \leq \mathbf{Adv}_{\mathsf{H}}^{\mathrm{COL}}(\lambda) + \frac{q_D}{q}, \tag{13}$$

where q_D denotes the number of the queries to the decapsulation oracle.

Proof of Claim (2). Suppose that $\mathcal{A}$ submits an invalid ciphertext $\psi = (u_1, u_2, v) \neq \psi^*$ to the decapsulation oracle. First, note that it is not possible that $(u_1, u_2) = (u_1^*, u_2^*)$ since $\psi \neq \psi^*$, we have $v \neq v^*$ and hence the decapsulation oracle will reject ψ straight away. Note also that it is possible that $(u_1, u_2) \neq (u_1^*, u_2^*)$ and $\alpha = \alpha^*$ *but* the probability that this happens is bounded by the insecurity of the hash function H since this event implies the violation of the target collision-resistance of H.

Thus, for up to q_D invalid ciphertexts such that $(u_1, u_2) \neq (u_1^*, u_2^*)$, we have $\alpha \neq \alpha^*$. In this case, if the point (x_1, x_2, y_1, y_2) lied on the hyperplane defined by the following equation

$$\log_{g_1} v = r_1 x_1 + r_2 x_2 w + r_1 y_1 \alpha + r_2 y_2 w \alpha, \tag{14}$$

where $r_1 = \log_{g_1} u_1$ and $r_2 = \log_{g_1} u_2$, the decapsulation oracle would accept the ciphertext ψ. However, observe that

$$\det \begin{bmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ r_1^* & r_2^* w & r_1^* \alpha^* & r_2^* \alpha^* w \\ r_1 & r_2 w & r_1 \alpha & r_2 \alpha w \end{bmatrix} = w^2 (r_1 - r_2)(r_1^* - r_2^*)(\alpha^* - \alpha) \neq 0.$$

Hence, (2), (3), (10) and (14) are linearly independent, implying that the hyperplane defined by (14) intersects the line formed by intersecting (2), (3) and (10) at a point, which happens with negligible probability $1/q$. Considering that there are q_D decapsulation queries, we get (13). This is the end of proof of *Claim (2)*.

Note that from (7) and (13), we get

$$\begin{aligned} \Pr[\beta' = \beta] &= \Pr[\beta' = \beta | \mathsf{RejInvC}] \Pr[\mathsf{RejInvC}] + \Pr[\beta' = \beta | \neg\mathsf{RejInvC}] \Pr[\neg\mathsf{RejInvC}] \\ &\leq \Pr[\beta' = \beta | \mathsf{RejInvC}] + \Pr[\neg\mathsf{RejInvC}] \\ &\leq \frac{1}{2} + \mathbf{Adv}^{\mathrm{ROR}}_{\mathsf{KDF}}(\lambda) + \mathbf{Adv}^{\mathrm{COL}}_{\mathsf{H}}(\lambda) + \frac{q_D}{q}. \end{aligned}$$

(The above inequality shows that regardless of the quantity of $\Pr[\beta'{=}\beta | \neg\mathsf{RejInvC}]$, i.e. the advantage that the adversary may get through querying invalid ciphertexts to the decapsulation oracle, $\Pr[\beta' = \beta]$ is not much deviated from $1/2$ due to $\Pr[\beta' = \beta | \neg\mathsf{RejInvC}]$ and $\Pr[\neg\mathsf{RejInvC}]$, which turn out to be negligible.)

Then, from the construction of $\mathcal{D}$, we have

$$\Pr[\mathcal{D}(1^\lambda, g_1, g_2, g_1^{r_1^*}, g_2^{r_2^*}) = 1] = \Pr[\beta' = \beta] \leq \frac{1}{2} + \mathbf{Adv}^{\mathrm{ROR}}_{\mathsf{KDF}}(\lambda) + \mathbf{Adv}^{\mathrm{COL}}_{\mathsf{H}}(\lambda) + \frac{q_D}{q}.$$

Combining the bounds from Lemmas 1 and 2 (i.e., by subtracting (1) from (6)), we get the bound in the theorem statement.

4 Comparisons

In Table 1, we summarize the basic parameters such as public key, ciphertext of CS-KEM [10], KD-KEM [17], HK-KEM [14] and ours. We also summarize whether those schemes provide full CCA-security, assuming the hardness of the DDH problem. Note that KD-KEM and HK-KEM are proven to be CCCA-secure [14], which is weaker than full CCA. Note also that it is an open problem to prove or disprove that HK-KEM provides full CCA-security.

As one can notice from the above table, our scheme is more efficient than the CS-KEM scheme while it is less efficient than the KD-KEM and HK-KEM schemes. However, an advantage of our scheme and CS-KEM schemes might be the simplicity that they provide full CCA-security without introducing additional primitive like MAC. – As formally shown in [3], one can generically convert a CCCA-secure KEM into a CCA-secure KEM by authenticating the CCCA-secure KEM ciphertext using a MAC. Hence, KD-KEM and HK-KEM

Table 1. Comparison of Our KEM Scheme with Other KEM Schemes

Scheme	Public key	Ciphertext	Key	Full CCA
CS-KEM [10]	g_1, g_2, c, d, h	$g_1^r, g_2^r, (cd^\alpha)^r$	$\mathsf{KDF}(g_1^r, h^r)$	Yes
KD-KEM [17]	g_1, g_2, c, d	g_1^r, g_2^r	$(cd^\alpha)^r$	No
HK-KEM [14]	g_1, c, d, h	$g_1^r, (cd^\alpha)^r$	h^r	Not Known
Ours	g_1, g_2, c, d	$g_1^r, g_2^r, (cd^\alpha)^r$	$\mathsf{KDF}(g_1^r, c^r)$	Yes

Table 2. Comparison of Computational Costs

Scheme	Enc. Cost	Dec. Cost
CS-KEM [10]	3E + 1DE (4.39E)	2DE (2.78E)
KD-KEM [17]	2E + 1DE (3.39E)	1DE (1.39E)
HK-KEM [14]	2E + 1DE (3.39E)	1SE (≈1.39E)
Ours	4E	2.78E

can be made to be CCA-secure by introducing the overhead of MAC. In this case, expansion of the ciphertext is unavoidable and as a result, the length of the ciphertext is close to the original CS-KEM and ours.

In Table 2, we summarize the computational costs of the above-mentioned schemes. In the table, "E" stands for "Exponentiation", "DE" stands for "Double Exponentiation", which is a special case of multi-exponentiation for two bases, e.g. $A^a B^b$, and finally "SE" stands for "Sequential Exponentiation" [7], which is as efficient as multi-exponentiation (in our case, double exponentiation).

Since there are many factors that determine the running time of various multi-exponentiation algorithms [2,19], it would be difficult to state decisively one double exponentiation is equivalent to how much of single exponentiation. (Note that if we use the naive approach that computes two single exponentiations separately and multiply them together, 1 DE = 2 E.) But if one adopts the "multi-exponentiation with a sliding window" algorithm assuming the unsigned binary representation of exponents as described in [2], one can obtain 1 DE = 1.39E if window size = 2 and the bit-length of $q = 256$. The figures in the parentheses in Table 2 are obtained based on this assumption.

Notice from the above table that in terms of computational costs, the difference between our scheme and both KD-KEM and HK-KEM is *less than one exponentiation.*

We also remark that as done for CS-KEM and KD-KEM respectively in [10] and in [21], one can make the key generation and decapsulation algorithms of our KEM scheme more efficient, which is described in detail in Appendix A.

5 Conclusion

In this paper, we proposed a new variant of the Cramer-Shoup KEM (CS-KEM) scheme which is more efficient than the original Cramer-Shoup KEM and fully CCA-secure in the standard model, relative to the DDH problem. Our result shows that the original CS-KEM can further be optimized without losing full CCA-security.

It is natural to ask whether the same technique (that is, to "reuse" d^r in $(c^\alpha d)^r$) can be applied to the dual version of KD-KEM scheme presented in [14]. We found that it is difficult to provide a security reduction in this case since, when an inconsistent ciphertext is queried to the decapsulation oracle, one cannot always extract a key uniformly distributed in the simulation.

Thus, an interesting open problem is how to construct a PKE scheme that is more efficient than the PKE schemes based on the KD-KEM or the dual KD-KEM.

Acknowledgement. The authors are grateful to Eike Kiltz for his suggestions of improvement on an earlier version of this paper. The authors also thank the anonymous referees of ACNS '09 for their valuable comments. The first, third and fourth authors are partially supported by the European Union project SMEPP-033563. The second author is partially supported by ARC Discovery Grant DP0877123.

References

1. Abe, M., Genaro, R., Kurosawa, K.: Tag-KEM/DEM: A New Framework for Hybrid Encryption and A New Analysis of Kurosawa-Desmedt KEM, Cryptology ePrint Archive, Report 2005/027 (2005) (Last update: 11 October 2006)
2. Avanzi, R.M.: The Complexity of Certain Multi-Exponentiation Techniques in Cryptography. Journal of Cryptology 18(4), 357–373 (2005)
3. Baek, J., Galindo, D., Susilo, W., Zhou, J.: Constructing Strong KEM from Weak KEM (or How to Revive the KEM/DEM Framework). In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 358–374. Springer, Heidelberg (2008)
4. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
5. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations Among Notions of Security for Public-Key Encryption Schemes. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 26–45. Springer, Heidelberg (1998)
6. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: ACM-CCS 1993, pp. 62–73. ACM Press, New York (1993)
7. Bernstein, D.J.: Pippenger's Exponentiation Algorithm (preprint) (2002), http://cr.yp.to
8. Cash, D., Kiltz, E., Shoup, V.: The Twin Diffie-Hellman Problem and Applications. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 127–145. Springer, Heidelberg (2008); full version available on Cryptology ePrint Archive: Report 2008/067
9. Cramer, R., Shoup, V.: A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
10. Cramer, R., Shoup, V.: Design and Analysis of Practical Public-key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. SIAM Journal of Computing 33, 167–226 (2003)

11. Dolev, D., Dwork, C., Naor, M.: Non-malleable Cryptography. In: STOC 1991, pp. 542–552. ACM Press, New York (1991)
12. Gennaro, R., Shoup, V.: A Note on An Encryption Scheme of Kurosawa and Desmedt, Cryptology ePrint Archive, Report 2004/294 (2004)
13. Herranz, J., Hofheinz, D., Kiltz, E.: The Kurosawa-Desmedt Key Encapsulation is not Chosen-Ciphertext Secure,Cryptology ePrint Archive, Report 2006/207 (2006)
14. Hofheinz, D., Kiltz, E.: Secure hybrid encryption from weakened key encapsulation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 553–571. Springer, Heidelberg (2007)
15. ISO 18033-2, An Emerging Standard for Public-Key Encryption (2004)
16. Kiltz, E.: Chosen-Ciphertext Secure Key-Encapsulation Based on Gap Hashed Diffie-Hellman. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 282–297. Springer, Heidelberg (2007)
17. Kurosawa, K., Desmedt, Y.: A New Paradigm of Hybrid Encryption Scheme. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 426–442. Springer, Heidelberg (2004)
18. Lu, X., Lai, X., He, D.: Improved efficiency of Kiltz07-KEM, Cryptology ePrint Archive, Report 2008/312 (2008)
19. Möller, B., Rupp, A.: Faster Multi-Exponentiation through Caching: Accelerating (EC)DSA Signature Verification. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 39–56. Springer, Heidelberg (2008)
20. Naor, M., Yung, M.: Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In: STOC 1990, pp. 427–437. ACM Press, New York (1990)
21. Phong, L.T., Ogata, W.: On Some Variations of Kurosawa-Desmedt Public-Key Encryption Scheme. IEICE Transactions 90-A(1), 226–230 (2007)
22. Shoup, V.: A Proposal for an ISO Standard for Public Key Encryption (version 2.1), ISO/IEC JTC 1/SC 27 (2001)

A An Efficient Variant of Our KEM Scheme

Description. Adopting the techniques in [10,21], one can design an efficient variant of our KEM scheme, which we denote by "$\widetilde{\Pi}$", as follows.

Key Generation: Pick a group $\mathbb{G}$ of prime order q and generator g_1 of $\mathbb{G}$. Pick a target-collision resistant hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_q^*$ and a key derivation function KDF. Then choose $(w, x, y) \in \mathbb{Z}_q^3$ at random and compute

$$g_2 = g_1^w,\ c = g_1^x,\ d = g_1^y.$$

Return public key $pk = (\mathbb{G}, q, g_1, g_2, c, d, \mathsf{H}, \mathsf{KDF})$ and private key $sk = (pk, x, y, w)$.
Encapsulation: Pick $r \in \mathbb{Z}_q^*$ at random. Compute

$$u_1 = g_1^r,\ u_2 = g_2^r,\ \alpha = \mathsf{H}(u_1, u_2),\ v = c^r d^{r\alpha},\ K = \mathsf{KDF}(u_1, c^r).$$

Return ciphertext $\psi = (u_1, u_2, v)$ and key K.
Decapsulation: Upon receiving $\psi = (u_1, u_2, v)$, compute

$$\alpha = \mathsf{H}(u_1, u_2),\ u_2' = u_1^w\ , v' = u_1^{x+y\alpha},\ K = \mathsf{KDF}(u_1, u_1^x).$$

If $u_2' = u_2$ and $v' = v$ then return K; otherwise, return $\perp$.

The above scheme is also CCA-secure. Regarding this, we prove the following theorem.

Theorem 2. *If the KEM scheme Π (described in Section 3) is CCA-secure then the above KEM scheme $\widetilde{\Pi}$ is CCA-secure. More precisely, we have*

$$\mathbf{Adv}_{\widetilde{\Pi}}^{CCA}(\lambda) \leq \mathbf{Adv}_{\Pi}^{CCA}(\lambda) + \frac{q_D}{q}.$$

where λ denotes the security parameter and q_D is the number of queries to the decapsulation oracle.

Proof. Fix an attacker $\mathcal{A}$ for the scheme Π. Also, fix an attacker $\tilde{\mathcal{A}}$ for the scheme $\widetilde{\Pi}$.

Assume that $\mathcal{A}$ is provided with the public key $pk = (\mathbb{G}, q, g_1, g_2, c, d)$ and the private key $sk = (pk, x_1, x_2, y_1, y_2)$, where g_1 and g_2 are generators of $\mathbb{G}$ and $c = g_1^{x_1} g_2^{x_2}$ and $d = g_1^{y_1} g_2^{y_2}$. $\mathcal{A}$ simply gives $\tilde{\mathcal{A}}$ pk as the public key of the scheme Π. $\mathcal{A}$ sets $g_2 = g_1^w$ for some $w \in \mathbb{Z}_q$, $x = x_1 + wx_2$ and $y = y_1 + wy_2$. (Note that $\mathcal{A}$ does not the value w.) Since $c = g_1^{x_1} g_2^{x_2} = g_1^{x_1+wx_2} = g^x$, $d = g_1^{y_1} g_2^{y_2} = g_1^{y_1+wy_2} = g^x$ by definition of w and (x, y), the public key pk is distributed identically in both $\mathcal{A}$ and $\tilde{\mathcal{A}}$'s view.

When $\tilde{\mathcal{A}}$ queries a ciphertext $\psi = (u_1, u_2, v)$ to the decapsulation oracle in the find stage, $\mathcal{A}$ forwards it to its decapsulation oracle, gets a decapsulation result and sends it back to $\tilde{\mathcal{A}}$.

Sometime later, $\mathcal{A}$ gets a challenge ciphertext and a key pair $(\psi^* = (u_1^*, u_2^*, v^*), K_\beta)$, where $\beta \in \{0, 1\}$ is chosen at random, and forwards the pair to $\tilde{\mathcal{A}}$ as a challenge ciphertext of the scheme $\widetilde{\Pi}$ and a key.

When $\tilde{\mathcal{A}}$ queries a ciphertext $\psi = (u_1, u_2, v)$ to the decapsulation oracle in the guess stage, $\mathcal{A}$ forwards it to its decapsulation oracle, gets a decapsulation result and sends it back to $\tilde{\mathcal{A}}$.

When $\tilde{\mathcal{A}}$ outputs its guess, $\mathcal{A}$ outputs it as its guess.

We compute the probability that an invalid ciphertext $\psi = (u_1, u_2, v)$, which should have been rejected, is accepted by the simulated decapsulation oracle.

Since we have assumed that $\psi = (u_1, u_2, v)$ is invalid, the condition $[(u_1^w \neq u_2) \wedge (u_1^{x+y\alpha} = v)]$ or $[(u_1^w = u_2) \wedge (u_1^{x+y\alpha} \neq v)]$ or $[(u_1^w \neq u_2) \wedge (u_1^{x+y\alpha} \neq v)]$ holds. However, if the last two conditions held, the simulated decapsulation oracle would reject ψ. Hence the first condition $[(u_1^w \neq u_2) \wedge (u_1^{x+y\alpha} = v)]$ must hold when invalid ψ is not rejected by the simulated decapsulation oracle. Note that $u_1^w \neq u_2$ means $r_1 \neq r_2$ where $r_1 = \log_{g_1} u_1$ and $r_2 = \log_{g_1} u_2$. Note also that since $x = x_1 + wx_2$ and $y = x_y + wy_2$, $u_1^{x+y\alpha} = v$ is equivalent to

$$\begin{aligned}
& [r_1\{(x_1 + wx_2) + (y_1 + wy_2)\alpha\}] \bmod q \\
= \ & [r_1(x_1 + y_1\alpha) + r_2 w(x_2 + y_2\alpha)] \bmod q \\
\Longleftrightarrow \ & w(r_1 - r_2)(x_1 + wy_2) = 0 \bmod q.
\end{aligned}$$

As $r_1 \neq r_2$ by the assumption and $w \neq 0 \bmod q$, the above equation holds with probability $1/q$, which is negligible. Hence we get the bound in the theorem statement.

An Efficient Identity-Based Online/Offline Encryption Scheme[⋆]

Joseph K. Liu and Jianying Zhou

Institute for Infocomm Research
Singapore
{ksliu,jyzhou}@i2r.a-star.edu.sg

Abstract. In this paper, we present an efficient Identity-based Online / Offline Encryption (IBOOE) scheme. An IBOOE scheme allows one to split the encryption into two phases. In the offline phase, most heavy computations such as exponentiation or pairing, if any, are done in this phase. Yet it does not require the knowledge of the plaintext or the receiver's identity. This nice property allows it can be executed 'offline', or inside some powerful device. The next phase is called the online phase, where only light computations such as integer addition, multiplication or hashing are needed, together with the plaintext and the receiver's identity. This can be executed inside some embedded device such as smart card or wireless sensor where the computation power is very limited. We propose an efficient IBOOE scheme, with great improvement in the computation requirement of both the offline, online encryption phase and decryption phase, together with much shorten ciphertext over previous schemes. Our scheme can be proven secure in the random oracle model.

1 Introduction

The notion of "online / offline" cryptographic algorithm was first introduced by Even, Goldreich and Micali [5], in the context of digital signature. With this notion, the signing process can be divided into two phases. The first phase is called *offline* phase which is executed prior to the arrival of a message and the second phase is called *online* phase which is performed after knowing the message. The online phase should be very fast and require only very light computation, such as integer multiplication or hashing. Other heavier computation such as exponentiation should be avoided in the online phase. In this way, online / offline schemes are particularly useful for low-power devices such as smartcard or wireless sensor applications. Those heavy computations are done in the offline phase which can be carried out by other powerful devices.

In parallel to online/offline signature [10,8,4,7], the first online/offline encryption scheme was first proposed by Guo, Mu and Chen [6]. Note that there is a slight difference in the definition between online/offline signature and encryption scheme. If we split the encryption process in the same way as the signing process

[⋆] The work in this paper is funded by the A*STAR project SEDS-0721330047.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 156–167, 2009.

(that is, put all heavy computation into the offline phase), it is trivial to separate some standard encryption, such as ElGamal encryption scheme. However, it is only suitable for the situation where the sender knows the recipient of the encrypted message in the offline phase, since the offline phase requires the knowledge of the public key of the recipient. We are not interested in this scenario. Instead, we consider a notion that allows the knowledge of the recipient is yet unknown in the offline phase. [6] uses this definition for their scheme, in the context of identity-based encryption.

There are some scenarios that may require the above online/offline encryption. Suppose there are some sensitive data stored in a smartcard, which has only very limited computation power. In order to send the sensitive data to a recipient in a secure way, it should be encrypted using the recipient's public key or identity. To ensure timely and efficient delivery, it would be much better if part of the encryption process could be done *prior* to knowing the data to be encrypted *and* the recipient's public key or identity.

Wireless sensor network (WSN) can be another situation where online/offline encryption is useful. Similar to smartcard, wireless sensor also has only limited resource. It may take very long time, or even impossible to execute heavy computation. Yet the data they collect may be sensitive which is necessary to be encrypted before sending back to the base stations. By using online/offline encryption, the offline part (containing all heavy computation) can be done by a third party at the setup or manufacturing stage. Obviously at this stage nothing is collected. Sometimes even the base station identity maybe still unknown to the wireless sensor. Online/offline encryption is a good solution in this scenario.

Identity-Based (ID-Based) Cryptosystem, introduced by Shamir [9], eliminates the necessity for checking the validity of certificates in traditional public key infrastructure (PKI). In an ID-based cryptosystem, public key of each user is easily computable from an arbitrary string corresponding to this user's identity (e.g. an email address, a telephone number, etc.). Using its master key, a private key generator (PKG) then computes a private key for each user. This property avoids the requirement of using certificates and associates implicitly a public key (i.e. user identity) to each user within the system. One only needs to know the recipient's identity in order to send an encrypted message to him. It avoids the complicated and costly certificate (chain) verification for the authentication purpose. In contrast, the traditional PKI needs an additional certification verification process, which is equivalent to the computation of *two* signature verifications.

Identity-based system is particularly suitable for power constrained device such as WSN or smartcard. The absence of certificate eliminates the costly certificate verification process. In addition, when there is a new node added to the network, other nodes do not need to have its certificate verified in order to communicate in a secure and authenticated way. This can greatly reduce communication overhead and computation cost.

1.1 Contribution

In this paper, we propose an efficient identity-based online/offline encryption (IBOOE) scheme. There are only two IBOOE schemes existed in the literature, both are proposed by Guo, Mu and Chen in [6]. Although they satisfy the definitions and basic requirements of an IBOOE, they are not actually very efficient. The first scheme (denoted by GMC-1) requires 7 pairing operations in the decryption stage. While for the second scheme (denoted by GMC-2), the ciphertext is very large (more than 6400 bits). Our proposed scheme provides a much better efficiency: We just require 2 pairing operations in the decryption stage. The ciphertext is only 1280 bits which is 40% smaller than GMC-1 and 80% smaller than GMC-2. Besides, our scheme requires lighter computation in both offline and online stage than both GMC-1 and GMC-2. Our scheme can be proven secure in the random oracle model.

1.2 Organization

The rest of our paper is organized as follow. Some definitions will be given in Section 2. We present our scheme in Section 3. It is followed by the detail comparison between our scheme and other schemes in Section 4. Finally we conclude the paper in Section 5.

2 Definitions

2.1 Pairings

We briefly review the bilinear pairing. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order q. Let P be a generator of $\mathbb{G}$, and e be a bilinear map such that $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ with the following properties:

1. *Bilinearity*: For all $U, V \in \mathbb{G}$, and $a, b \in \mathbb{Z}$, $e(aU, bV) = e(U, V)^{ab}$.
2. *Non-degeneracy*: $e(P, P) \neq 1$.
3. *Computability*: It is efficient to compute $e(U, V)$ for all $U, V \in \mathbb{G}$.

2.2 Intractability Assumption

Definition 1 (ℓ-Bilinear Diffie-Hellman Inversion Assumption (ℓ-BDHI)). [2] *The ℓ- Diffie-Hellman (ℓ-BDHI) problem in $\mathbb{G}$ is defined as follow: On input a $(\ell+1)$-tuple $(P, \alpha P, \alpha^2 P, \cdots, \alpha^\ell P) \in \mathbb{G}^{\ell+1}$, output $e(P, P)^{\frac{1}{\alpha}} \in \mathbb{G}_T$. We say that the (t, ϵ, ℓ)-BDHI assumption holds in $\mathbb{G}$ if no t-time algorithm has advantage at least ϵ in solving the ℓ-BDHI problem in $\mathbb{G}$.*

2.3 Definition of ID-Based Online/Offline Encryption

An ID-based online/offline encryption scheme consists of the following five probabilistic polynomial time (PPT) algorithms:

- $(param, msk) \leftarrow \mathsf{Setup}(1^k)$ takes a security parameter $k \in \mathbb{N}$ and generates $param$ the global public parameters and msk the master secret key of the PKG.

- $D_{ID} \leftarrow \mathsf{Extract}(1^k, param, msk, ID)$ takes a security parameter k, a global parameters $param$, a master secret key msk and an identity ID to generate a secret key D_{ID} corresponding to this identity.
- $\bar{\phi} \leftarrow \mathsf{Offline\text{-}encrypt}(1^k, param)$ takes a security parameter k and a global parameters $param$ to generate an offline ciphertext $\bar{\phi}$.
- $\phi \leftarrow \mathsf{Online\text{-}encrypt}(1^k, param, m, \bar{\phi}, ID)$ takes a security parameter k, a global parameters $param$, a message m, an offline ciphertext $\bar{\phi}$, an identity ID to generate a ciphertext ϕ.
- $m/\perp \leftarrow \mathsf{Decrypt}(1^k, param, \phi, D_{ID})$ takes a security parameter k, a global parameters $param$, a ciphertext ϕ, a secret key of the receiver D_{ID} to generate a message m or $\perp$ which indicates the failure of decryption.

For simplicity, we omit the notation of 1^k and $param$ from the input arguments of the above algorithms in the rest of this paper.

2.4 Security of ID-Based Online/Offline Encryption

Definition 2 (Chosen Ciphertext Security). *An ID-based online/offline encryption scheme is semantically secure against chosen ciphertext insider attack (ID-IND-CCA) if no PPT adversary has a non-negligible advantage in the following game:*

1. *The challenger runs* Setup *and gives the resulting param to adversary $\mathcal{A}$. It keeps msk secret.*
2. *In the first stage, $\mathcal{A}$ makes a number of queries to the following oracles which are simulated by the challenger:*
   - *(a)* ***Extraction oracle****: $\mathcal{A}$ submits an identity ID to the extraction oracle for the result of $\mathsf{Extract}(msk, ID)$.*
   - *(b)* ***Decryption oracle****: $\mathcal{A}$ submits a ciphertext ϕ and a receiver identity ID to the oracle for the result of $\mathsf{Decrypt}(\phi, D_{ID})$. The result is made of a message if the decryption is successful. Otherwise, a symbol $\perp$ is returned for rejection.*

   These queries can be asked adaptively. That is, each query may depend on the answers of previous ones.
3. *$\mathcal{A}$ produces two messages m_0, m_1 and an identities ID^*. The challenger chooses a random bit $b \in \{0,1\}$ and computes an encrypted ciphertext $\phi^* = \mathsf{Online\text{-}Encrypt}(m_b,\ \mathsf{Offline\text{-}Encrypt}(), ID^*)$. ϕ^* is sent to $\mathcal{A}$.*
4. *$\mathcal{A}$ makes a number of new queries as in the first stage with the restriction that it cannot query the decryption oracle with (ϕ^*, ID^*) and the extraction oracle with ID^*.*
5. *At the end of the game, $\mathcal{A}$ outputs a bit b' and wins if $b' = b$.*

$\mathcal{A}$'s advantage is defined as $\boldsymbol{Adv}^{IND-CCA}(\mathcal{A}) = |\Pr[b' = b] - \frac{1}{2}|$.

3 The Proposed Online/Offline ID-Based Encryption Scheme

3.1 Construction

Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups of prime-order q, and let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be the bilinear pairing. We use a multiplicative notation for the operation in $\mathbb{G}$ and $\mathbb{G}_T$.

Setup: The PKG selects a generator $P \in \mathbb{G}$ and randomly chooses $s, w \in_R \mathbb{Z}_q^*$. It sets $P_{pub} = sP$, $P'_{pub} = s^2P$ and $W = (w+s)^{-1}P$. Define $\mathcal{M}$ to be the message space. Let $n_M = |\mathcal{M}|$. Also let $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$, $H_2 : \{0,1\}^* \times \mathbb{G}_T \rightarrow \mathbb{Z}_q^*$ and $H_3 : \{0,1\}^* \rightarrow \{0,1\}^{n_M}$ be some cryptographic hash functions. The public parameters *param* and master secret key *msk* are given by

$$param = (\mathbb{G}, \mathbb{G}_T, q, P, P_{pub}, P'_{pub}, W, w, \mathcal{M}, H_1, H_2, H_3) \qquad msk = s$$

Extract: To generate a secret key for a user with identity $ID \in \{0,1\}^{n_d}$, the PKG computes:

$$D_{ID} = (H_1(ID) + s)^{-1} P$$

Offline-Encrypt: Randomly generates $u, x, \alpha, \beta, \gamma, \delta \in_R \mathbb{Z}_q^*$ and computes:

$$U \leftarrow W - uP \qquad R \leftarrow e(wP + P_{pub}, P)^x$$

$$T_0 \leftarrow x\Big(w\alpha P + \big(w + \gamma\big)P_{pub} + P'_{pub}\Big)$$

$$T_1 \leftarrow xw\beta P \qquad T_2 \leftarrow x\delta P_{pub}$$

Outputs the offline ciphertext $\bar{\phi} = (u, x, \alpha, \beta, \gamma, \delta, U, R, T_0, T_1, T_2)$. Note that $e(wP + P_{pub}, P)$ can be pre-computed by the PKG as part of the *param* so that no pairing is needed in this phase.

Online-Encrypt: To encrypt a message $m \in \mathcal{M}$ to ID, at the online stage, computes:

$$t'_1 \leftarrow \beta^{-1}\Big(H_1(ID) - \alpha\Big) \bmod q \qquad t'_2 \leftarrow \delta^{-1}\Big(H_1(ID) - \gamma\Big) \bmod q$$

$$t \leftarrow H_2(m, R)x + u \bmod q \qquad c \leftarrow H_3(R) \oplus m$$

Outputs the ciphertext $\phi = (U, T_0, T_1, T_2, t'_1, t'_2, t, c)$.

Decrypt: To decrypt using secret key D_{ID}, computes

$$R \leftarrow e(T_0 + t'_1T_1 + t'_2T_2\ ,\ D_{ID}) \qquad m \leftarrow c \oplus H_3(R)$$

and checks whether

$$R^{H_2(m,R)} \overset{?}{=} e\big(tP + U\ ,\ wP + P_{pub}\big) \cdot e(P,P)^{-1} \tag{1}$$

Outputs m if it is equal. Otherwise outputs $\perp$.

Same as above, $e(P,P)$ can be pre-computed by the PKG as part of the *param*.

3.2 Security Analysis

Correctness. For the decrypt, we have

$$\begin{aligned}
&e(T_0 + t'_1 T_1 + t'_2 T_2, D_{ID})\\
&= e\Big(x\Big(wH_1(ID)P + wP_{pub} + H_1(ID)P_{pub} + P'_{pub}\Big),\\
&\quad \Big(H_1(ID) + s\Big)^{-1} P\Big)\\
&= e\Big(x\Big(w\Big(H_1(ID) + s\Big)P + \Big(H_1(ID) + s\Big)P_{pub}\Big),\\
&\quad \Big(H_1(ID) + s\Big)^{-1} P\Big)\\
&= e\Big(x\Big(\Big(H_1(ID) + s\Big)\Big(wP + P_{pub}\Big)\Big), \Big(H_1(ID) + s\Big)^{-1} P\Big)\\
&= e\Big(x\Big(\Big(H_1(ID) + s\Big)\Big(w + s\Big)P\Big), \Big(H_1(ID) + s\Big)^{-1} P\Big)\\
&= e\Big((w + s)P, P\Big)^{x}\\
&= R
\end{aligned}$$

On the other side, let $h = H_2(m, R)$. We have

$$\begin{aligned}
&e\Big(tP + U, wP + P_{pub}\Big) \cdot e(P,P)^{-1}\\
&= e\Big(hxP + uP + \Big(w + s\Big)^{-1} P - uP, \Big(w + s\Big)P\Big) \cdot e(P,P)^{-1}\\
&= e\Big(hxP + \Big(w + s\Big)^{-1} P, \Big(w + s\Big)P\Big) \cdot e(P,P)^{-1}\\
&= e\Big(hxP, \Big(w + s\Big)P\Big) \cdot e\Big(\Big(w + s\Big)^{-1} P, \Big(w + s\Big)P\Big) \cdot e(P,P)^{-1}\\
&= \Big(e\Big(P, wP + P_{pub}\Big)^{x}\Big)^{h} = R^h
\end{aligned}$$

Theorem 1. *If there is a ID-IND-CCA adversary $\mathcal{A}$ of the proposed scheme that succeeds with probability ϵ, then there is a simulator $\mathcal{B}$ running in polynomial time that solves the $(\ell + 1)$-BDHI problem with probability at least*

$$\epsilon \cdot \frac{1}{q_1}\Big(1 - \frac{q_d}{q}\Big)$$

where q_1, q_d are the number of queries allowed to the random oracle H_1 and decryption oracle respectively and we assume $q_1 = \ell$.

Proof. Setup: We follow the proof technique from [1]. Suppose $\mathcal{B}$ is given a random instance of the $(\ell+1)$-BDHI problem $(\hat{P}, \alpha\hat{P}, \alpha^2\hat{P}, \ldots, \alpha^\ell\hat{P}, \alpha^{\ell+1}\hat{P})$, $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine to output $e(\hat{P},\hat{P})^{\frac{1}{\alpha}}$. $\mathcal{B}$ sets up a simulated environment for $\mathcal{A}$ as follow.

$\mathcal{B}$ first randomly selects $\pi \in_R \{1,\ldots,q_1\}$, $I_\pi \in_R \mathbb{Z}_q^*$ and $w_1,\ldots,$ $w_{\pi-1}$, $w_{\pi+1},\ldots,w_\ell \in_R \mathbb{Z}_q^*$. For $i \in \{1,\ldots,\ell\}\setminus\{\pi\}$, it computes $I_i = I_\pi - w_i$. Construct a polynomial with degree $\ell-1$ as

$$f(z) = \prod_{i=1, i\neq\pi}^{\ell} (z + w_i)$$

to obtain $c_0,\ldots,c_{\ell-1} \in \mathbb{Z}_q^*$ such that $f(z) = \sum_{i=0}^{\ell-1} c_i z^i$. Then it sets generator $G = \sum_{i=0}^{\ell-1} c_i(\alpha^i\hat{P}) = f(\alpha)\hat{P}$.

For $i \in \{1,\ldots,\ell\}\setminus\{\pi\}$, $\mathcal{B}$ expands $f_i(z) = f(z)/(z+w_i) = \sum_{j=0}^{\ell-2} d_{i,j}z^j$ to obtain $d_{i,1},\ldots,d_{i,\ell-2} \in \mathbb{Z}_q^*$ and sets

$$\tilde{H}_i = \sum_{j=0}^{\ell-2} d_{i,j}(\alpha^j\hat{P}) = f_i(\alpha)\hat{P} = \frac{f(\alpha)}{\alpha+w_i}\hat{P} = \frac{1}{\alpha+w_i}G$$

It randomly chooses $\hat{w} \in \{1,\ldots,\ell\}\setminus\{\pi\}$, and computes the public key w, W, P_{pub} and P'_{pub} as

$$w = I_{\hat{w}} \qquad W = -\tilde{H}_{\hat{w}}$$

$$P_{pub} = -\alpha G - I_\pi G = (-\alpha - I_\pi)G \qquad P'_{pub} = \alpha^2 G + 2I_\pi\alpha G + I_\pi^2 G = (\alpha + I_\pi)^2 G$$

where $\alpha G = \sum_{i=0}^{\ell-1} c_i(\alpha^{i+1}\hat{P})$ and $\alpha^2 G = \sum_{i=0}^{\ell-1} c_i(\alpha^{i+2}\hat{P})$ so that its unknown master secret key msk is implicitly set to $x = -\alpha - I_\pi \in \mathbb{Z}_q^*$, while public parameter $param$ are set to $(G, P_{pub}, P'_{pub}, W, w)$ which are given to the adversary. For all $i \in \{1,\ldots,\ell\}\setminus\{\pi\}$, we have $(I_i, -\tilde{H}_i) = (I_i, \frac{1}{I_i+x}G)$.

Oracle Simulation: $\mathcal{B}$ first initializes a counter ν to 1 and starts $\mathcal{A}$. Throughout the game, we assume that H_1-queries are distinct, that the target identity ID^* is submitted to H_1 at some point.

1. *Random Oracle:* For H_1-queries (we denote ID_ν the input of the ν^{th} one of such queries), $\mathcal{B}$ answers I_ν and increments ν.
   For H_2-queries on input (m, R) and H_3-queries on input R, $\mathcal{B}$ returns the defined value if it exists and a randomly chosen $h_2 \in_R \mathbb{Z}_q^*$ for H_2 and $h_3 \in_R \{0,1\}^{n_m}$ for H_3 respectively, otherwise. $\mathcal{B}$ stores the information $(m, R, h_2, c = m \oplus h_3, \gamma = R^{h_2} \cdot e(G,G))$ in L_2 and (R, h_3) in L_3.
2. *Extraction Oracle:* On input ID_ν, if $\nu = \pi$, $\mathcal{B}$ aborts. Otherwise, it knows that $H_1(ID_\nu) = I_\nu$ and returns $-\tilde{H}_\nu = (1/(I_\nu + x))G$.
3. *Decryption Oracle:* On input a ciphertext $\phi = (U, T_0, T_1, T_2, t'_1, t'_2, t, c)$ for identity ID_ν, we assume that $\nu = \pi$ because otherwise $\mathcal{B}$ knows the receiver's

private key $D_{ID_\nu} = -\tilde{H}_\nu$ and can normally run the decryption algorithm. Let $\tilde{x} \in \mathbb{Z}_q$ such that

$$\begin{aligned}\tilde{x}W &= tG + U - W\\ \tilde{x}(w+x)^{-1}G &= tG + U - (w+x)^{-1}G\\ \tilde{x}G &= (w+x)(tG+U) - G \qquad (2)\end{aligned}$$

Also let $T = T_0 + t'_1T_1 + t'_2T_2$, $G_{ID_\nu} = I_\nu G + P_{pub}$, and $h = H_2(m, R)$ (which is yet unknown to $\mathcal{B}$ at this moment). As all valid ciphertext satisfies

$$\begin{aligned}R^h &= e(tG+U, (w+x)G)\cdot e(G,G)^{-1}\\ e(hT, (I_\nu + x)^{-1}G) &= e((w+x)(tG+U), G)\cdot e(-G,G)\\ e((I_\nu + x)^{-1}hT, G) &= e((w+x)(tG+U) - G, G)\\ (I_\nu + s)^{-1}hT &= (w+x)(tG+U) - G \qquad (3)\end{aligned}$$

Let $\tilde{x}' \in \mathbb{Z}_q$ such that

$$\begin{aligned}\tilde{x}'G_{ID_\nu} &= hT\\ \tilde{x}'(I_\nu + x)G &= hT\\ \tilde{x}'G &= (I_\nu + x)^{-1}hT\\ &= (w+x)(tG+U) - G \text{ (from equation(3))}\\ &= \tilde{x}G \text{ (from equation(2))}\\ \Rightarrow \tilde{x}' &= \tilde{x}\\ \Rightarrow \log_W(tG+U-W) &= \log_{G_{ID_\nu}}(hT) \qquad (4)\end{aligned}$$

From equation (4), we have

$$e(hT, W) = e(G_{ID_\nu}, S - W) \qquad (5)$$

where $S = tG + U$, which yields $e(hT, W) = e(G_{ID_\nu}, S)\cdot e(G_{ID_\nu}, W)^{-1}$.

The query is handled by computing $\gamma = e(S, wG + P_{pub})$, and search through the list L_2 for entries of the form $(m_i, R_i, h_{2,i}, c, \gamma)$ indexed by $i \in \{1, \ldots, q_2\}$. If none is found, ϕ is rejected. Otherwise, each one of them is further examined: for the corresponding indexes, $\mathcal{B}$ checks if

$$\frac{e(T,W)^{h_{2,i}}}{e(S, G_{ID_\nu})} = e(G_{ID_\nu}, W)^{-1} \qquad (6)$$

meaning that equation (5) is satisfied. If the unique $i \in \{1, \ldots, q_2\}$ satisfying equation (6) is detected, the matching pair $(m_i, h_{2,i}, S)$ is returned. Otherwise ϕ is rejected.

Challenge: $\mathcal{A}$ outputs messages (m_0, m_1) and identities ID^* for which it never obtained ID^*'s private key. If $ID^* \neq ID_\pi$, $\mathcal{B}$ aborts. Otherwise it randomly selects $t, t'_1, t'_2, \tilde{t}_0, \tilde{t}_1, \tilde{t}_2 \in_R \mathbb{Z}_q^*$, $c \in_R \{0,1\}^{n_m}$ and $U \in_R \mathbb{G}$. Computes $T_0 = \tilde{t}_0 G, T_1 = \tilde{t}_1 G, T_2 = \tilde{t}_2 G$ to return the challenge ciphertext $\phi^* = (U, t, T_0, T_1, T_2, t'_1, t'_2, c)$. Let $\xi = \tilde{t}_0 + t'_1 \tilde{t}_1 + t'_2 \tilde{t}_2$ and $T = -\xi G$. Since $x = -\alpha - I_\pi$, we let $\rho = \frac{\xi}{\alpha(w-\alpha-I_\pi)} = -\frac{\xi}{(I_\pi+x)(w+x)}$, we can check that

$$\begin{aligned} T &= -\xi G = -\alpha(w - \alpha - I_\pi)\rho G \\ &= (I_\pi + x)(w + x)\rho G \\ &= \rho(I_\pi w + (w + I_\pi)x + x^2)G \end{aligned}$$

$\mathcal{A}$ cannot recognize that ϕ^* is not a proper ciphertext unless it queries H_2 or H_3 on $e(wG + G_{pub}, G)^\rho$. Along the guess stage, its view is simulated as before and its output is ignored. Standard arguments can show that a successful $\mathcal{A}$ is very likely to query H_2 or H_3 on the input $e(G_{ID_\mu}, G)^\rho$ if the simulation is indistinguishable from a real attack environment.

Output Calculation: $\mathcal{B}$ fetches a random entry (m, R, h_2, c, γ) or $(R, \cdot)$ from the lists L_2 or L_3. With probability $1/(2q_2 + q_3)$, the chosen entry will contain the right element

$$R = e(wG + P_{pub}, G)^\rho = e(G, G)^{-\xi/(I_\pi + x)} = e(\hat{P}, \hat{P})^{f(\alpha)^2 \xi/\alpha}$$

where $f(z) = \sum^* \ell - 1_{i=0} c_i z^i$ is the polynomial for which $G = f(\alpha)P$. The $(\ell + 1)$-BDHI solution can be extracted by computing

$$\begin{aligned} &\left(\frac{R^{1/\xi}}{e\Big(\sum_{i=0}^{\ell-2} c_{i+1}(\alpha^i \hat{P}), c_0 \hat{P} \Big) e\Big(\sum_{j=0}^{\ell-2} c_{j+1}(\alpha^j) \hat{P}, G \Big)} \right)^{1/c_0^2} \\ &= \left(\frac{e(\hat{P}, \hat{P})^{f(\alpha)^2/\alpha}}{e(\hat{P}, \hat{P})^{c_0(c_1 + c_2\alpha + c_3\alpha^2 + \ldots c_{\ell-1}\alpha^{\ell-2})} e(\hat{P}, \hat{P})^{f(\alpha)(c_1 + c_2\alpha + c_3\alpha^2 + \ldots c_{\ell-1}\alpha^{\ell-2})}} \right)^{1/c_0^2} \\ &= \left(\frac{e(\hat{P}, \hat{P})^{f(\alpha)^2/\alpha}}{e(\hat{P}, \hat{P})^{\frac{c_0(c_1\alpha + c_2\alpha^2 + \ldots c_{\ell-1}\alpha^{\ell-1}) + f(\alpha)(c_1\alpha + c_2\alpha^2 + \ldots c_{\ell-1}\alpha^{\ell-1})}{\alpha}}} \right)^{1/c_0^2} \\ &= e(\hat{P}, \hat{P})^{\frac{f(\alpha)^2 - (c_1\alpha + c_2\alpha^2 + \ldots c_{\ell-1}\alpha^{\ell-1})(c_0 + f(\alpha))}{{c_0}^2 \alpha}} \\ &= e(\hat{P}, \hat{P})^{\frac{{c_0}^2}{{c_0}^2 \alpha}} \\ &= e(\hat{P}, \hat{P})^{1/\alpha} \end{aligned}$$

Probability Analysis: $\mathcal{B}$ only fails in providing a consistent simulation because one of the following independent events happen:

- E_1 : $\mathcal{A}$ does not choose to be challenged on ID_π.
- E_2 : A key extraction query is made on ID_π.
- E_3 : $\mathcal{B}$ rejects a valid ciphertext at some point of the game.

We have $\Pr[\neg E_1] = 1/q_1$ and $\neg E_1$ implies $\neg E_2$. Also observe that $\Pr[E_3] \leq q_d/q$. Combining together, the overall successful probability $\Pr[\neg E_1 \wedge \neg E_3]$ is at least

$$\frac{1}{q_1}\left(1 - \frac{q_d}{q}\right)$$

□

4 Comparison

There are only 2 existing online/offline IBE schemes, both of them are proposed by Guo, Mu and Chen in [6]. We use GMC-1 and GMC-2 to denote them. We also assume that $|\mathbb{G}| = 160$ bits, $|q| = 160$ bits, $|\mathbb{G}_T| = 1024$ bits and $|\mathcal{M}| = |q| = 160$ bits [1] for the following comparison. We denote by E the point multiplication in $\mathbb{G}$ or $\mathbb{G}_T$, ME the multi-point multiplication in $\mathbb{G}$ or $\mathbb{G}_T$ (which costs about 1.3 times more than a single point multiplication), M the point addition in $\mathbb{G}$ or $\mathbb{G}_T$ and m_c the modular computation in $\mathbb{Z}_q$.

Table 1. Comparison of computation cost and size

	GMC-1	GMC-2	Our scheme
Offline computation	$5E + 2ME$	$4E + 2ME$	$4E + 1ME$
Online computation	$1M + 2m_c$	$1M + 2m_c$	$3m_c$
Offline storage (bits)	2624	5056	2624
Ciphertext length (bits)	2144	6464	1280
Number of pairing for decryption	7	2	2
Security model	selective ID	standard	random oracle

We note that as GMC-1 requires an online/offline signature for encryption, we assume they use the most efficient one [3] which requires 320 bits for offline storage and 320 bits for signature length. The key generation and offline signing part require 1 E operation respectively. These costs have been added to the table.

From the above table, we can see that our scheme achieves the least computation and the smallest size in both offline and online stage, when compare to GMC-1 and GMC-2. There are a number of significant improvements:

1. First, we do not require any point addition operation (M operation) in the online encryption stage. Modular computation (m_c operation) is much faster than M operation. Other computations that required in our scheme such as hashing or XOR are negligible when compared to M operation. Thus our online encryption stage is much faster than GMC-1 and GMC-2.

[1] In our scheme, the message space can be any arbitrary length but the message space of GMC-1 and GMC-2 can be only set to the size of q, the order of group $\mathbb{G}_T$. In order to compare three schemes, we also set the message space of our scheme to 160 bits.

2. Second, the offline storage is as small as GMC-1 and about 50% smaller than GMC-2. This result is important in embedded device such as smart card or wireless sensor, where the storage is very limited.
3. Third, the ciphertext of our scheme is 40% smaller than GMC-1 and 80% smaller than GMC-2. This improvement is very significant in the environment where the communication bandwidth is very limited. On the other side, the number of pairing operations required in the decryption stage, is just 2, which is the same as GMC-2 while 3 times less than GMC-1. In other words, we combine and improve the efficiency advantages of both GMC-1 (short ciphertext) and GMC-2 (small number of pairing operations in decryption).
4. Forth, our scheme allows the message space to be any arbitrary length while the message space of GMC-1 and GMC-2 should be equal to the size of q, the order of group $\mathbb{G}_T$. Usually q is chosen as a 160-bits prime. That means the message space of GMC-1 and GMC-2 is 160 bits. If a larger message, say 1024 bits, is encrypted using GMC-1 or GMC-2, it must be divided into 7 parts ($\lceil \frac{1024}{160} \rceil = 7$) and carried out the encryption process 7 times. However in our scheme we just need to adjust the output length of the hash function H_3 to be 1024 and increase the size of the ciphertext from 1280 to 2144 bits ($1280 + (1024 - 160) = 2144$). Then we only need to execute the whole encryption process *once* (instead of 7 times, when compared to GMC-1 and GMC-2). Our scheme is particularly useful for a large message space.

We also remark that our scheme can be proven secure in the random oracle model, which is relatively stronger than the standard model or the selective-ID model. Although it is generally believed that random oracle model is not as secure as standard model theoretically, it still achieves an acceptable level of security. There are many applications that put efficiency as the most important factor. In these scenarios, schemes that are efficient but can be only proven secure in the random oracle model maybe a better choice.

5 Conclusion

In this paper we have proposed a new efficient identity-based online/offline encryption scheme. When compared to previous schemes, our scheme enjoys a number of significant improvements in efficiency. These improvements allow our scheme to be used in many practical scenarios such as smart card and wireless sensor networks. Our scheme can be proven secure in the random oracle model.

References

1. Barreto, P., Libert, B., McCullagh, N., Quisquater, J.: Efficient and provabley-secure identity-based signature and signcryption from bilinear maps. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 515–532. Springer, Heidelberg (2005)

2. 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)
3. Boneh, D., Boyen, X.: Short signatures without random oracles the SDH assumption in bilinear groups. Journal of Cryptology 2, 149–177 (2008)
4. Chen, X., Zhang, F., Susilo, W., Mu, Y.: Efficient generic online/offline signatures without key exposure. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 18–30. Springer, Heidelberg (2007)
5. Even, S., Goldreich, O., Micali, S.: On-line/offline digital signatures. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 263–277. Springer, Heidelberg (1990)
6. Guo, F., Mu, Y., Chen, Z.: Identity-based online/offline encryption. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 247–261. Springer, Heidelberg (2008)
7. Joye, M.: An efficient on-line/off-line signature scheme without random oracles. In: Franklin, M.K., Hui, L.C.K., Wong, D.S. (eds.) CANS 2008. LNCS, vol. 5339, pp. 98–107. Springer, Heidelberg (2008)
8. Kurosawa, K., Schmidt-Samoa, K.: New online/offline signature schemes without random oracles. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 330–346. Springer, Heidelberg (2006)
9. 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)
10. Shamir, A., Tauman, Y.: Improved online/offline signature schemes. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 355–367. Springer, Heidelberg (2001)

Dual-Policy Attribute Based Encryption

Nuttapong Attrapadung and Hideki Imai

Research Center for Information Security (RCIS),
National Institute of Advanced Industrial Science and Technology (AIST)
Akihabara-Daibiru Room 1003, 1-18-13, Sotokanda,
Chiyoda-ku, Tokyo 101-0021 Japan
{n.attrapadung,h-imai}@aist.go.jp

Abstract. We present a new variant of Attribute based encryption (ABE) called Dual-Policy ABE. Basically, it is a conjunctively combined scheme between Key-Policy and Ciphertext-Policy ABE, the two previous available types of ABE. Dual-Policy ABE allows *simultaneously* two access control mechanisms over encrypted data: one involves policies over *objective* attributes ascribed to data and the other involves policies over *subjective* attributes ascribed to user credentials. The previous two types of ABE can only allow either functionality above one at a time.

Keywords: Attribute-based encryption, Ciphertext policy, Key policy.

1 Introduction

Attribute-based encryption (ABE) enables an access control mechanism over encrypted data using access policies and ascribed attributes among private keys and ciphertexts. ABE comes in two flavors called Ciphertext-Policy ABE and Key-Policy ABE.

In Ciphertext-Policy ABE, an encryptor can express any access policy, stating what kind of receivers will be able to decrypt the message, directly in the encryption algorithm (which can be run by anyone knowing the universal public key issued priorly by an authority). Such a policy is specified in terms of access structure over attributes. A user is ascribed by an attribute set, in the sense that each attribute corresponds to one of her credential, and is priorly given the private key from the authority. Such a user can decrypt a ciphertext if her attribute satisfies the access policy associated to the ciphertext. An example application of CP-ABE is secure mailing list system with access policy. There, a private key will be assigned for an attribute set, such as {"MANAGER", "AGE:30", "INSTITUTE:ABC"}, while policies over attributes such as "MANAGER" $\vee$ ("TRAINEE" $\wedge$ "AGE:25") will be associated to ciphertexts.

In Key-Policy ABE, the roles of an attribute set and an access policy are swapped from what we described for CP-ABE. Attribute sets are used to annotate the ciphertexts and access policies over these attributes are associated to users' secret keys. An example application of KP-ABE is Pay-TV system with package policy (called target broadcast system in [6]). There, a ciphertext

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 168–185, 2009.

will associate with an attribute set, such as {"TITLE:24", "GENRE:SUSPENSE", "SEASON:2", "EPISODE:13" }, while policies over attributes such as "SOCCER" $\vee$ ("TITLE:24" $\wedge$ "SEASON:5") will be associated to TV program package keys that user receives when subscribes.

A drawback of the above two previous types of ABE is that we must choose whether attributes will be used to annotate either the ciphertexts, which we call *objects* (since they are to be decrypted), or the users' credentials, which we call *subjects* (since users are to decrypt); after setup we must also stick with such condition throughout the entire application. To see why this is inconvenient, we give an example in the Pay-TV application above. Since we are using KP-ABE, the encrypted movie can only be ascribed by *objective* attributes. Thus, the broadcast station, which is the encryptor, cannot directly specify *subjective* access policy, *i.e.,* who can or cannot decrypt. It might want to do so, since it may want, for example, to directly include or revoke some user credentials. The same inconvenience happens also for CP-ABE complimentarily.

In this paper, we present a new type of ABE called Dual-Policy ABE, which resolves the above problem affirmatively. Basically, it is a conjunctively combined scheme between KP and CP ABE. Dual-Policy ABE works as follows. An encryptor can associate the data simultaneously with both a set *objective* attributes that annotate the data itself and a *subjective* access policy that states what kind of receivers will be able to decrypt. On the other hand, a user is given a private key assigned simultaneously for both a set of *subjective* attributes that annotate user's credentials and a *subjective* access policy that states what kind of data she can decrypt. The decryption can be done if and only if the objective attribute set satisfies the objective policy *and* the subjective attribute set satisfies the subjective policy.

Previous Works. ABE was introduced by Sahai and Waters [9] in the context of a generalization of ID-based encryption (IBE) [2] called Fuzzy IBE, which is an ABE that allows only single threshold access structures. The first (and still being state-of-the-art) KP-ABE scheme that allows any monotone access structures was proposed by Goyal et al. [6], while the first such CP-ABE scheme, albeit with the security proof in the generic bilinear group model, was proposed by Bethencourt, Sahai, and Waters [1]. Ostrovsky, Sahai, and Waters [8] then subsequently extended both to handle also any non-monotone structures. Goyal et al. [5] presented bounded CP-ABE in the standard model. Waters [10] recently proposed the first fully expressive CP-ABE in the standard model.

Our Approach. Our DP-ABE scheme is based on an algebraic combination of CP-ABE by Waters [10] and KP-ABE by Goyal et al. [6]. We note that such a combination is non-trivial at the first place, since, for example, one may think of obtaining DP-ABE by using AND-double encryption (even in a secure way) of KP-ABE and CP-ABE. However, one can easily find out that this mislead method is insecure due to collusion attacks. Our scheme utilizes more sophisticated techniques for secure integration.

Our DP-ABE subsumes both KP-ABE and CP-ABE in the sense that when neglecting objective attributes our scheme becomes CP-ABE of Waters [10] and when neglecting subjective attributes it becomes KP-ABE of Goyal et al. [6].

Furthermore, our DP-ABE scheme also realizes the delegation of private keys. An interesting property is that we can also delegate the key of pure KP-ABE to a key of DP-ABE, where subjective attribute dimension is added, and the key of pure CP-ABE to a key of DP-ABE, where objective attribute dimension is added. Therefore our DP-ABE scheme is extended seamlessly from both KP-ABE of [6] and CP-ABE of [10].

Another feature of our DP-ABE scheme is that even such a scheme has been already set-up to be used as DP-ABE, it can also be used as if it were KP-ABE or CP-ABE on-the-fly by using encryption in what we call single-policy modes. This flexibility provides great convenience since the same instantiated key can be used for all three variants of ABE.

More Related Works. Recently, Boneh and Hamburg [3] formalized a very general framework called Generalized IBE (GIBE) which also includes both of ABE variants as special cases. DP-ABE also falls into their framework: it can be casted as a product scheme between KP-ABE and CP-ABE. However, their instantiated construction for KP-ABE seems to have large key size that is linear to the access structure collection size, which could be super-polynomially large.

Another similar general framework called predicate encryption was proposed previously by Katz, Sahai, and Waters [7]. Their system achieves also anonymity property, where the information about access structures or attribute sets associated with ciphertexts itself is kept hidden. However, their system tends to handle only less expressive access structures than systems without anonymity.

Organization of the Paper. We first provide preliminary materials such as the notion of linear secret sharing and bilinear pairing in Section 2. We then present the definition and the security notion of Dual-Policy ABE in Section 3. In Section 4, we present our concrete DP-ABE scheme called DPABE. In Section 5, we describe the key delegation of our DP-ABE scheme. In Section 6, we present both generic and specific enhanced schemes for DP-ABE that admit single-policy modes. We then conclude in Section 7. The security proofs of the schemes with key delegation and single-policy modes are given in Appendix A.1,A.2.

2 Preliminaries

We first provide the notion of access structure and linear secret sharing scheme as follows. Such formalization is recapped from [10].

Definition 1 (Access Structure). *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\mathcal{P}}$ is monotone if for all B, C we have that if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (respectively, monotonic access structure) is a collection (respectively, monotone collection) $\mathbb{A} \subseteq 2^{\mathcal{P}} \setminus \{\emptyset\}$.*

Definition 2 (Linear Secret Sharing Schemes (LSSS)). *Let $\mathcal{P}$ be a set of parties. Let M be a matrix of size $\ell \times k$. Let $\rho : \{1, \ldots, \ell\} \to \mathcal{P}$ be a function that maps a row to a party for labeling. A secret sharing scheme Π for access structure $\mathbb{A}$ over a set of parties $\mathcal{P}$ is a linear secret-sharing scheme in $\mathbb{Z}_p$ and is represented by (M, ρ) if it consists of two polynomial-time algorithms:*

$\mathsf{Share}_{(M,\rho)}$: *The algorithm takes as input $s \in \mathbb{Z}_p$ which is to be shared. It randomly chooses $y_2, \ldots, y_k \in \mathbb{Z}_p$ and let $\boldsymbol{v} = (s, y_2, \ldots, y_k)$. It outputs $M\boldsymbol{v}$ as the vector of ℓ shares. The share $\lambda_{\rho(i)} := \boldsymbol{M_i} \cdot \boldsymbol{v}$ belongs to party $\rho(i)$, where we denote $\boldsymbol{M_i}$ as the ith row in M.*

$\mathsf{Recon}_{(M,\rho)}$: *The algorithm takes as input $S \in \mathbb{A}$. Let $I = \{i|\ \rho(i) \in S\}$. It outputs reconstruction constants $\{(i, \mu_i)\}_{i \in I}$ which has a linear reconstruction property: $\sum_{i \in I} \mu_i \cdot \lambda_{\rho(i)} = s$.*

Proposition 1. *Let (M, ρ) be a LSSS for access structure $\mathbb{A}$ over a set of parties $\mathcal{P}$, where M is a matrix of size $\ell \times k$. For all $S \notin \mathbb{A}$, there exists a polynomial time algorithm that outputs a vector $\boldsymbol{w} = (w_1, \ldots, w_k) \in \mathbb{Z}_p^k$ such that $w_1 = -1$ and for all $x \in S$ it holds that $\boldsymbol{M_i} \cdot \boldsymbol{w} = 0$.*

Bilinear Maps. We briefly review facts about bilinear maps. Let $\mathbb{G}, \mathbb{G}_T$ be multiplicative groups of prime order p. Let g be a generator of $\mathbb{G}$. A bilinear map is a map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ for which the following hold: (1) e is bilinear; that is, for all $u, v \in \mathbb{G}$, $a, b \in \mathbb{Z}$, we have $e(u^a, v^b) = e(u, v)^{ab}$. (2) The map is non-degenerate: $e(g, g) \neq 1$. We say that $\mathbb{G}$ is a bilinear group if the group action in $\mathbb{G}$ can be computed efficiently and there exists $\mathbb{G}_T$ for which the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is efficiently computable.

Decision BDHE Assumption. Let $\mathbb{G}$ be a bilinear group of prime order p. The Decision q-BDHE (Bilinear Diffie-Hellman Exponent) problem [4] in $\mathbb{G}$ is stated as follows: given a vector

$$\Big(g, h, g^{\alpha}, g^{(\alpha^2)}, \ldots, g^{(\alpha^q)}, g^{(\alpha^{q+2})}, \ldots, g^{(\alpha^{2q})}, Z\Big)$$

$\in \mathbb{G}^{2q+1} \times \mathbb{G}_T$ as input, determine if $Z = e(g, h)^{(\alpha^{q+1})}$. We denote $g_i = g^{(\alpha^i)} \in \mathbb{G}$ for shorthand. Let $\boldsymbol{y}_{g,\alpha,q} = (g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q})$. An algorithm $\mathcal{A}$ that outputs $b \in \{0, 1\}$ has advantage ϵ in solving Decision q-BDHE in $\mathbb{G}$ if

$$|\Pr\big[\mathcal{A}\big(g, h, \boldsymbol{y}_{g,\alpha,q}, e(g_{q+1}, h)\big) = 0\big] - \Pr\big[\mathcal{A}\big(g, h, \boldsymbol{y}_{g,\alpha,q}, Z\big) = 0\big]| \geq \epsilon,$$

where the probability is over the random choice of generators $g, h \in \mathbb{G}$, the random choice of $\alpha \in \mathbb{Z}_p$, the random choice of $Z \in \mathbb{G}_T$, and the randomness of $\mathcal{A}$. We refer to the distribution on the left as $\mathcal{P}_{BDHE}$ and on the right as $\mathcal{R}_{BDHE}$. We say that the Decision q-BDHE assumption holds in $\mathbb{G}$ if no polynomial-time algorithm has a non-negligible advantage in solving the problem.

3 Definitions

A Dual-policy attribute-based encryption scheme consists of four algorithms.

Setup: This is a randomized algorithm that takes no input other than the implicit security parameter. It outputs public key pk and master key msk.

$\mathsf{Encrypt}(\mathsf{pk}, \mathcal{M}, (\mathbb{S}, \omega))$: This is a randomized algorithm that takes as input the public key pk, a message $\mathcal{M}$, a subjective access structure $\mathbb{S}$, a set of objective attributes ω. It outputs the ciphertext ct.

$\mathsf{KeyGen}(\mathsf{pk}, \mathsf{msk}, (\psi, \mathbb{O}))$: This is a randomized algorithm that takes as input the public key pk, the master key msk, a set of subjective attributes ψ, an objective access structure $\mathbb{O}$. It outputs a private decryption key sk.

$\mathsf{Decrypt}(\mathsf{pk}, (\psi, \mathbb{O}), \mathsf{sk}, (\mathbb{S}, \omega), \mathsf{ct})$: This algorithm takes as input the public key pk, a decryption key sk and its associated pair of set of subjective attributes ψ and objective access structure $\mathbb{O}$, a ciphertext ct and its associated pair of subjective access structure $\mathbb{S}$ and set of objective attributes ω. It outputs the message $\mathcal{M}$ if it holds that the set ω of objective attributes satisfies the objective access structure $\mathbb{O}$ and that the set ψ of subjective attributes satisfies the subjective access structure $\mathbb{S}$, *i.e., $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$.*

We require the standard correctness of decryption: if $\mathsf{Setup} \rightarrow (\mathsf{pk}, \mathsf{msk})$ then $\mathsf{Decrypt}\Big(\mathsf{pk}, (\psi, \mathbb{O}), \mathsf{KeyGen}(\mathsf{pk}, \mathsf{msk}, (\psi, \mathbb{O})), (\mathbb{S}, \omega), \mathsf{Encrypt}(\mathsf{pk}, \mathcal{M}, (\mathbb{S}, \omega))\Big) \rightarrow \mathcal{M}$, for all $\mathcal{M}$ in the message space and all $\omega \in \mathbb{O}$ and $\psi \in \mathbb{S}$.

The selective security notion for DP-ABE is defined in the following game.

Init. The adversary declares the target subjective access structure $\mathbb{S}^\star$ and the target objective attribute set $\omega^\star$.

Setup. The challenger runs the Setup algorithm of DP-ABE and gives the public key pk to the adversary.

Phase 1. The adversary is allowed to issue queries for private keys for pairs of subjective attribute set and objective access structure $(\psi, \mathbb{O})$ such that $\omega^\star \notin \mathbb{O}$ *or* $\psi \notin \mathbb{S}^\star$, *i.e.,* the negated condition of that of a legitimate key which can be used to decrypt a challenge ciphertext.

Challenge. The adversary submits two equal length messages $\mathcal{M}_0$ and $\mathcal{M}_1$. The challenger flips a random bit b and computes the challenge ciphertext $\mathsf{ct}^\star$ on the target pair $(\mathbb{S}^\star, \omega^\star)$ of subjective access structure and objective attribute set and then gives $\mathsf{ct}^\star$ to the adversary.

Phase 2. Phase 1 is repeated.

Guess. The adversary outputs a guess b' of b.

The advantage of an adversary in this game is defined as $\Pr[b = b'] - \frac{1}{2}$. Note that this can be extended to handle chosen-ciphertext attacks by allowing decryption queries in Phase 1,2.

Definition 3. *A DP-ABE scheme is secure in the selective-set security notion if all polynomial time adversaries have at most a negligible advantage in the above game.*

4 Dual-Policy ABE Scheme

Our DP-ABE scheme will be based on a combination of CP-ABE by Waters [10] and KP-ABE by Goyal et al. [6]. Both subjective and objective access structures are those which there exist linear secret sharing schemes that realize them. We denote by (M, ρ) a LSSS scheme that represents a subjective access structure $\mathbb{S}$ and by (N, π) a LSSS scheme that represents a objective access structure $\mathbb{O}$. We will restrict ρ to be an injective function as in Waters [10] scheme, but we can extend to an unrestricted scheme, also similarly as in [10].

4.1 Main Construction

Let m be the maximum size of subjective attribute set allowed to be assigned to a key, *i.e.,* we restrict $|\psi| \leq m$. Let n be the maximum size of objective attribute set allowed to be associated with a ciphertext, *i.e.,* we restrict $|\omega| \leq n$. Let $\ell_{\mathsf{s,max}}$ be the maximum number of rows allowed in a subjective access structure matrix. Let $m' = m + \ell_{\mathsf{s,max}} - 1$ and $n' = n - 1$. Our main scheme DPABE is described as follows. Let $\mathcal{U}_{\mathsf{s}}, \mathcal{U}_{\mathsf{o}}$ be the universe of subjective and objective attributes, respectively.

▶ Setup: The algorithm first picks a random generator $g \in \mathbb{G}$ and random exponent $\gamma, a \in \mathbb{Z}_p$. It then defines two functions $F_{\mathsf{s}} : \mathbb{Z}_p \to \mathbb{G}$ and $F_{\mathsf{o}} : \mathbb{Z}_p \to \mathbb{G}$ by first randomly choosing $h_0, \ldots, h_{m'}, t_0, \ldots, t_{n'} \in \mathbb{G}$ and setting

$$F_{\mathsf{s}}(x) = \prod_{j=0}^{m'} h_j^{x^j}, \quad F_{\mathsf{o}}(x) = \prod_{j=0}^{n'} t_j^{x^j}.$$

It assigns the public key as $\mathsf{pk} = (g, e(g,g)^{\gamma}, g^a, h_0, \ldots, h_{m'}, t_0, \ldots, t_{n'})$. The master key is $\mathsf{msk} = (\gamma, a)$.

▶ $\mathsf{Encrypt}$: Inputs to the encryption algorithm are a LSSS access structure (M, ρ) for subjective policy and a objective attribute set $\omega \subset \mathcal{U}_{\mathsf{o}}$. Let M be $\ell_{\mathsf{s}} \times k_{\mathsf{s}}$ matrix. The algorithm first randomly chooses $s, y_2, \ldots, y_{k_{\mathsf{s}}} \in \mathbb{Z}_p$ and lets $\boldsymbol{u} = (s, y_2, \ldots, y_{k_{\mathsf{s}}})$. For $i = 1$ to ℓ_{s}, it calculates $\lambda_i = \boldsymbol{M_i} \cdot \boldsymbol{u}$, where $\boldsymbol{M_i}$ is the vector corresponding to ith row of M. The ciphertext ct is set to $\mathsf{ct} = (C, \hat{C}, \{C_i\}_{i=1,\ldots,\ell_{\mathsf{s}}}, \{C'_x\}_{x \in \omega})$, where

$$C = \mathcal{M} \cdot (e(g,g)^{\gamma})^s, \qquad \hat{C} = g^s,$$
$$C_i = g^{a\lambda_i} F_{\mathsf{s}}(\rho(i))^{-s}, \qquad C'_x = F_{\mathsf{o}}(x)^s.$$

▶ KeyGen: Inputs to the encryption algorithm are a LSSS access structure (N, π) for objective policy and a subjective attribute set $\psi \subset \mathcal{U}_{\mathsf{s}}$. Let N be $\ell_{\mathsf{o}} \times k_{\mathsf{o}}$ matrix. The algorithm first randomly chooses $r, z_2 \ldots, z_{k_{\mathsf{o}}} \in \mathbb{Z}_p$ and lets $\boldsymbol{v} = (\gamma + ar, z_2, \ldots, z_{k_{\mathsf{o}}})$. For $i = 1$ to ℓ_{o}, it calculates $\sigma_i = \boldsymbol{N_i} \cdot \boldsymbol{v}$, where $\boldsymbol{N_i}$ is the vector corresponding to ith row of N. It also randomly chooses $r_1, \ldots, r_{\ell_{\mathsf{o}}} \in \mathbb{Z}_p$.

It creates the private decryption key as $\mathsf{sk} = (K, \{K_x\}_{x\in\psi}, \{\hat{K}_i, K'_i\}_{i=1,\ldots,\ell_\mathsf{o}})$, where

$$K = g^r, \qquad K_x = F_\mathsf{s}(x)^r,$$
$$\hat{K}_i = g^{\sigma_i} F_\mathsf{o}(\pi(i))^{-r_i}, \qquad K'_i = g^{r_i}.$$

▶ $\mathsf{Decrypt}$: The decryption algorithm takes as input the ciphertext ct which contains a subjective access structure (M, ρ) and a set of objective attributes ω, and a decryption key sk which contains a set of subjective attributes ψ and an objective access structure (N, π). Suppose that the set ψ for subjective attribute satisfies (M, ρ) and that the set ω for objective attribute satisfies (N, π) (so that the decryption is possible). We then let $I_\mathsf{s} = \{i|\ \rho(i) \in \psi\}$ and $I_\mathsf{o} = \{i|\ \pi(i) \in \omega\}$. It then calculates corresponding sets of reconstruction constants $\{(i, \mu_i)\}_{i\in I_\mathsf{s}} = \mathsf{Recon}_{(M,\rho)}(\psi)$ and $\{(i, \nu_i)\}_{i\in I_\mathsf{o}} = \mathsf{Recon}_{(N,\pi)}(\omega)$. The decryption algorithm then computes

$$C \cdot \frac{\prod_{i\in I_\mathsf{s}} \Big(e(C_i, K) \cdot e(\hat{C}, K_{\rho(i)})\Big)^{\mu_i}}{\prod_{j\in I_\mathsf{o}} \Big(e(\hat{K}_j, \hat{C}) \cdot e(K'_j, C'_{\pi(j)})\Big)^{\nu_j}} = \mathcal{M}. \tag{1}$$

Correctness. We verify the correctness of the decryption as follows. Let sk and ct be defined as in the scheme above. We first note that from linear reconstruction property of the LSSS schemes, we have

$$\sum_{i\in I_\mathsf{s}} \mu_i \lambda_i = s, \quad \sum_{i\in I_\mathsf{o}} \nu_i \sigma_i = \gamma + ar. \tag{2}$$

The correctness can then be verified as

$$\begin{aligned} & C \cdot \frac{\prod_{i\in I_\mathsf{s}} \Big(e(C_i, K) \cdot e(\hat{C}, K_{\rho(i)})\Big)^{\mu_i}}{\prod_{j\in I_\mathsf{o}} \Big(e(\hat{K}_j, \hat{C}) \cdot e(K'_j, C'_{\pi(j)})\Big)^{\nu_j}} \\ &= C \cdot \frac{\prod_{i\in I_\mathsf{s}} \Big(e(g^{a\lambda_i} F_\mathsf{s}(\rho(i))^{-s}, g^r) \cdot e(g^s, F_\mathsf{s}(\rho(i))^r)\Big)^{\mu_i}}{\prod_{j\in I_\mathsf{o}} \Big(e(g^{\sigma_j} F_\mathsf{o}(\pi(j))^{-r_j}, g^s) \cdot e(g^{r_j}, F_\mathsf{o}(\pi(j))^s)\Big)^{\nu_j}} \\ &= C \cdot \frac{\prod_{i\in I_\mathsf{s}} e(g^{a\lambda_i}, g^r)^{\mu_i}}{\prod_{j\in I_\mathsf{o}} e(g^{\sigma_j}, g^s)^{\nu_j}} = C \cdot \frac{e(g^{as}, g^r)}{e(g^{\gamma+ar}, g^s)} = C \cdot \frac{1}{e(g,g)^{\gamma s}} = \mathcal{M}. \end{aligned}$$

Remark 1. The above decryption algorithm of Eq.(1) was written only for ease of visualizing. A more efficient computation with the less number of applications of pairing can be done as follows. Note that Eq.(3) requires only $|\omega|+2$ applications of pairing, while Eq.(1) requires $2(|\omega| + |\psi|)$ such applications.

$$C \cdot \frac{e\Big(\big(\prod_{i\in I_\mathsf{s}} C_i^{\mu_i}\big), K\Big)}{\prod_{j\in I_\mathsf{o}} e\Big(K'_j, C'_{\pi(j)}\Big)^{\nu_j}} \cdot e\left(\hat{C}, \frac{\Big(\prod_{i\in I_\mathsf{s}} K_{\rho(i)}^{\mu_i}\Big)}{\Big(\prod_{j\in I_\mathsf{o}} \hat{K}_j^{\nu_j}\Big)}\right) = \mathcal{M}. \tag{3}$$

4.2 Security Proof

Theorem 1. *If an adversary can break the* DPABE *scheme with advantage ϵ in the selective-set security model for DP-ABE with a challenge subjective access structure matrix of size $\ell_{\mathsf{s}}^{\star} \times k_{\mathsf{s}}^{\star}$, then a simulator with advantage ϵ in solving the Decision q-BDHE problem can be constructed, where $m + k_{\mathsf{s}}^{\star} \leq q$.*

The proof follows mostly from [6,10] with some non-trivial adaptation mostly in simulating the private keys.

Proof. Suppose there exists an adversary, $\mathcal{A}$, that has advantage ϵ in attacking the DPABE scheme. We build a simulator $\mathcal{B}$ that solves the Decision q-BDHE problem in $\mathbb{G}$. $\mathcal{B}$ is given as input a random q-BDHE challenge $(g, h, \boldsymbol{y}_{g,\alpha,q}, Z)$, where $\boldsymbol{y}_{g,\alpha,q} = (g_1, \ldots, g_q, g_{q+2}, \ldots, g_{2q})$ and Z is either $e(g_{q+1}, h)$ or a random element in $\mathbb{G}_1$ (recall that $g_j = g^{(\alpha^j)}$). $\mathcal{B}$ proceeds as follows.

Init. The selective-set game begins with $\mathcal{A}$ first outputting $((M^{\star}, \rho^{\star}), \omega^{\star})$, where $(M^{\star}, \rho^{\star})$ is a target subjective access structure in the form of LSSS matrix and $\omega^{\star}$ is a target objective attribute set. Let $M^{\star}$ be of size $\ell_{\mathsf{s}}^{\star} \times k_{\mathsf{s}}^{\star}$, where $m + k_{\mathsf{s}}^{\star} \leq q$. Wlog, we can assume that $\ell_{\mathsf{s}}^{\star} = \ell_{\mathsf{s,max}}$ and $|\omega^{\star}| = n$.

Setup. $\mathcal{B}$ chooses random $\gamma' \in \mathbb{Z}_p$ and implicitly sets $\gamma = \gamma' + \alpha^{q+1}$ by letting $e(g,g)^{\gamma} = e(\alpha, \alpha^q) e(g,g)^{\gamma'}$. It also lets $g^a = g^{\alpha}$.

The simulator then programs the function F_{s} by defining $F_{\mathsf{s}}(x) = g^{p(x)}$, where p is a polynomial in $\mathbb{Z}_p[x]$ of degree $m + \ell_{\mathsf{s}}^{\star} - 1$ which is implicitly defined as follows. It first chooses $k_{\mathsf{s}}^{\star} + m + 1$ polynomial $p_0, \ldots, p_{k_{\mathsf{s}}^{\star}+m}$ in $\mathbb{Z}_p[x]$ of degree $m + \ell_{\mathsf{s}}^{\star} - 1$ in such a way that for x such that there exists an i where $x = \rho^{\star}(i)$ (there are exactly $\ell_{\mathsf{s}}^{\star}$ values of such x, since $\rho^{\star}$ is injective) we set

$$p_j(x) = \begin{cases} M_{i,j}^{\star} & \text{for } j \in [1, k_{\mathsf{s}}^{\star}], \\ 0 & \text{for } j \in [k_{\mathsf{s}}^{\star}+1, k_{\mathsf{s}}^{\star}+m], \end{cases}$$

and random for x elsewhere (by randomly picking values at some other m points for each polynomial) and p_0 is chosen completely randomly. Write coefficients in each polynomial as $p_j(x) = \sum_{i=0}^{m+\ell_{\mathsf{s}}^{\star}-1} p_{j,i} \cdot x^i$. It then conceptually defines

$$p(x) = \sum_{j=0}^{k_{\mathsf{s}}^{\star}+m} p_j(x) \cdot \alpha^j.$$

by setting $h_i = \prod_{j=0}^{k_{\mathsf{s}}^{\star}+m} g_j^{p_{j,i}}$ for $i \in [0, m + \ell_{\mathsf{s}}^{\star} - 1]$. From the definition of F_{s} in the scheme, one can verify that

$$F_{\mathsf{s}}(x) = \prod_{i=0}^{m+\ell_{\mathsf{s}}^{\star}-1} h_i^{x^i} = g^{p(x)}.$$

The simulator then programs the next function F_{o} as follows. It randomly picks a polynomial in $\mathbb{Z}_p[x]$ of degree $n-1$, $f'(x) = \sum_{j=0}^{n-1} f_j' x^j$. Next it defines

$f(x) = \prod_{k\in\omega^\star}(x-k) = \sum_{j=0}^{n-1} f_j x^j$. We note that f_j's terms can be computed completely from $\omega^\star$. From this we can ensure that $f(x)=0$ if and only if $x\in\omega^\star$. It then lets $t_j = g_q^{f_j} g^{f'_j}$ for $j=[0,n-1]$. We thus have

$$F_{\mathsf{o}}(x) = \prod_{j=0}^{n-1} t_j^{(x^j)} = g_q^{f(x)} \cdot g^{f'(x)}.$$

It then gives the public key $\mathsf{pk} = (g, e(g,g)^\gamma, g_1, h_0, \ldots, h_{m'}, t_0, \ldots, t_{n'})$ to $\mathcal{A}$.

Phase 1. The adversary makes requests for private keys corresponding to objective access structure and subjective attribute set pair $((N,\pi),\psi)$ subjected to condition that ψ does not satisfy $M^\star$ *or* $\omega^\star$ does not satisfy N. We distinguish two cases due to the latter condition.

[**Case 1**: $\omega^\star$ does not satisfy N].

The simulator randomly chooses $r\in\mathbb{Z}_p$. It then lets $K=g^r$ and for all $x\in\psi$ lets $K_x = F_{\mathsf{s}}(x)^r$ as in the construction.

Due to the condition in this case and by Proposition 1, there must exist a vector $\boldsymbol{a} = (a_1,\ldots,a_{k_{\mathsf{o}}}) \in \mathbb{Z}_p^{k_{\mathsf{o}}}$ such that $a_1=-1$ and that for all i where $\pi(i)\in\omega^\star$, it holds that $\boldsymbol{N_i}\cdot\boldsymbol{a}=0$.

The simulator randomly chooses $z'_2,\ldots,z'_{k_{\mathsf{o}}}\in\mathbb{Z}_p$ and lets $\boldsymbol{v'} = (0, z'_2,\ldots,z'_{k_{\mathsf{o}}})$. It implicitly defines a vector $\boldsymbol{v} = -(\gamma' + \alpha^{q+1} + \alpha r)\boldsymbol{a} + \boldsymbol{v'}$, which will be used for creating shares of $\gamma+\alpha r$ as in the construction.

For i where $\pi(i)\in\omega^\star$, it randomly chooses $r_i\in\mathbb{Z}_p$ and computes $K'_i = g^{r_i}$ and

$$\hat{K}_i = g^{\boldsymbol{N_i}\cdot\boldsymbol{v'}} F_{\mathsf{o}}(\pi(i))^{-r_i} = g^{\boldsymbol{N_i}\cdot\boldsymbol{v}} F_{\mathsf{o}}(\pi(i))^{-r_i},$$

where the right equality is due to $\boldsymbol{N_i}\cdot\boldsymbol{a}=0$.

For i where $\pi(i)\notin\omega^\star$, it randomly chooses $r'_i\in\mathbb{Z}_p$. Observe that

$$\boldsymbol{N_i}\cdot\boldsymbol{v} = (\boldsymbol{N_i}\cdot\boldsymbol{a})\alpha^{q+1} + (r\boldsymbol{N_i}\cdot\boldsymbol{a})\alpha + \boldsymbol{N_i}\cdot(\boldsymbol{v'} - \gamma'\boldsymbol{a})$$

contains the term α^{q+1}, thus we cannot compute $g^{\boldsymbol{N_i}\cdot\boldsymbol{v}}$ as usual. We will use the term $F_{\mathsf{o}}(\pi(i))^{-r_i}$ to cancel out the unknown value. To do this it implicitly defines $r_i = r'_i + \frac{\alpha(\boldsymbol{N_i}\cdot\boldsymbol{a})}{f(\pi(i))}$. This can be done by setting

$$\begin{aligned}\hat{K}_i &= g_1^{\left(r\boldsymbol{N_i}\cdot\boldsymbol{a} - \frac{(\boldsymbol{N_i}\cdot\boldsymbol{a})f'(\pi(i))}{f(\pi(i))}\right)} \cdot g^{\boldsymbol{N_i}\cdot(\boldsymbol{v'}-\gamma'\boldsymbol{a})} \cdot F_{\mathsf{o}}(\pi(i))^{-r'_i},\\ K'_i &= g^{r'_i} g_1^{\frac{(\boldsymbol{N_i}\cdot\boldsymbol{a})}{f(\pi(i))}} = g^{r_i},\end{aligned}$$

which can be computed since $\pi(i)\notin\omega^\star$ hence $f(\pi(i))\neq 0$. The correctness of $\hat{K}_i$ can be verified as:

$$\begin{aligned}\hat{K}_i &= (g_{q+1})^{\boldsymbol{N_i}\boldsymbol{a}} g_1^{r\boldsymbol{N_i}\cdot\boldsymbol{a}} g^{\boldsymbol{N_i}\cdot(\boldsymbol{v'}-\gamma'\boldsymbol{a})} \cdot \left((g_{q+1})^{-\boldsymbol{N_i}\cdot\boldsymbol{a}} g_1^{-\frac{(\boldsymbol{N_i}\cdot\boldsymbol{a})f'(\pi(i))}{f(\pi(i))}}\right) \cdot F_{\mathsf{o}}(\pi(i))^{-r'_i}\\ &= g^{\boldsymbol{N_i}\cdot\boldsymbol{v}} \cdot F_{\mathsf{o}}(\pi(i))^{-\frac{\alpha(\boldsymbol{N_i}\cdot\boldsymbol{a})}{f(\pi(i))}} \cdot F_{\mathsf{o}}(\pi(i))^{-r'_i} = g^{\boldsymbol{N_i}\cdot\boldsymbol{v}} \cdot F_{\mathsf{o}}(\pi(i))^{-r_i}.\end{aligned}$$

[**Case 2**: $\omega^\star$ satisfies N].

In this case, we must have that ψ does not satisfy $M^\star$. Therefore, by Proposition 1 and our definition of p_j above, there must exist a vector $(w_1, \dots, w_{k_\mathsf{s}^\star}) \in \mathbb{Z}_p^{k_\mathsf{s}^\star}$ such that $w_1 = -1$ and for all $x \in \psi$ such that there exist i where $x = \rho^\star(i)$, we have $(p_1(x), \dots, p_{k_\mathsf{s}^\star}(x)) \cdot (w_1, \dots, w_{k_\mathsf{s}^\star}) = 0$. Next it also computes one possible solution of variables $w_{k_\mathsf{s}^\star+1}, \dots, w_{k_\mathsf{s}^\star+m}$ from the system of $|\psi|$ equations: for all $x \in \psi$,

$$(p_1(x), \dots, p_{k_\mathsf{s}^\star+m}(x)) \cdot (w_1, \dots, w_{k_\mathsf{s}^\star+m}) = 0,$$

which is possible since $|\psi| \le m$. Now we define $\boldsymbol{b_x} = (p_1(x), \dots, p_{k_\mathsf{s}^\star+m}(x))$ and $\boldsymbol{w} = (w_1, \dots, w_{k_\mathsf{s}^\star+m})$. Thus, for all $x \in \psi$ we have $\boldsymbol{b_x} \cdot \boldsymbol{w} = 0$.

The simulator then randomly chooses $r' \in \mathbb{Z}_p$ and implicitly defining

$$r = r' + w_1 \cdot \alpha^q + w_2 \cdot \alpha^{q-1} + \cdots w_{k_\mathsf{s}^\star+m} \cdot \alpha^{q-(k_\mathsf{s}^\star+m)+1}, \tag{4}$$

by setting $K = g^{r'} \prod_{k=1}^{k_\mathsf{s}^\star+m} (g_{q+1-k})^{w_k} = g^r$. From our definition of r, we have

$$\gamma + \alpha r = \gamma' + \alpha r' + w_2 \alpha^q + \cdots + w_{k_\mathsf{s}^\star+m} \cdot \alpha^{q-(k_\mathsf{s}^\star+m)+2},$$

where we observe that the important term α^{q+1} in γ is canceled out. It randomly chooses $z_2 \dots, z_{k_\mathsf{o}} \in \mathbb{Z}_p$ and implicitly lets $\boldsymbol{v} = (\gamma + \alpha r, z_2, \dots, z_{k_\mathsf{o}})$ as in the construction. It also randomly chooses $r_1, \dots, r_{\ell_\mathsf{o}} \in \mathbb{Z}_p$. It then computes for $i = 1$ to ℓ_o, $K_i' = g^{r_i}$ and

$$\hat{K}_i = \big(g^{\gamma'} g_1^{r'} \prod_{k=2}^{k_\mathsf{s}^\star+m} (g_{q-k+2})^{w_k}\big)^{N_{i,1}} \cdot \prod_{j=2}^{k_\mathsf{o}} g^{N_{i,j} z_j} \cdot F_\mathsf{o}(\pi(i))^{-r_i},$$

where one can verify that $\hat{K}_i = g^{\boldsymbol{N_i} \cdot \boldsymbol{v}} \cdot F_\mathsf{o}(\pi(i))^{-r_i}$. We can compute this since g_{q+1} is not contained. The simulator then creates K_x for all $x \in \psi$ as:

$$K_x = K^{p_0(x)} \cdot \prod_{j=1}^{k_\mathsf{s}^\star+m} \left(g_j^{r'} \prod_{\substack{k \in [1, k_\mathsf{s}^\star+m] \\ k \neq j}} (g_{q+1-k+j})^{w_k} \right)^{p_j(x)},$$

where one can verify that $K_x = F_\mathsf{s}(x)^r$ by observing that since for all $x \in \psi$, we have $\boldsymbol{b_x} \cdot \boldsymbol{w} = 0$; therefore,

$$\begin{aligned} K_x = K_x \cdot (g_{q+1})^{\boldsymbol{b_x} \cdot \boldsymbol{w}} &= K_x \cdot \prod_{j=1}^{k_\mathsf{s}^\star+m} (g_{q+1-j+j})^{w_j p_j(x)} \\ &= K^{p_0(x)} \cdot \prod_{j=1}^{k_\mathsf{s}^\star+m} \left(g_j^{r'} \prod_{k=1}^{k_\mathsf{s}^\star+m} (g_{q+1-k+j})^{w_k} \right)^{p_j(x)} \\ &= (g^r)^{p_0(x)} \cdot \prod_{j=1}^{k_\mathsf{s}^\star+m} (g^r)^{\alpha^j p_j(x)} = (g^r)^{p(x)} = F_\mathsf{s}(x)^r. \end{aligned}$$

Challenge. The adversary gives two message $\mathcal{M}_0, \mathcal{M}_1$ to the simulator. The simulator flips a coin b and creates $C = \mathcal{M}_b \cdot Z \cdot e(h, g^{\gamma'})$, $\hat{C} = h$, and for $x \in \omega^\star$, $C'_x = h^{f'(x)}$. Write $h = g^s$ for some unknown s. The simulator chooses randomly $y'_2, \ldots, y_{k^\star_{\mathsf{s}}} \in \mathbb{Z}_p$. Let $\boldsymbol{y'} = (0, y'_2, \ldots, y_{k^\star_{\mathsf{s}}})$. It will then implicitly share the secret s using the vector

$$\boldsymbol{v} = (s, s\alpha + y'_2, s\alpha^2 + y'_3, \ldots, s\alpha^{k^\star_{\mathsf{s}}-1} + y'_{k^\star_{\mathsf{s}}}),$$

by setting for $i = 1, \ldots, \ell^\star_{\mathsf{s}}$, $C_i = (g_1)^{\boldsymbol{M^\star_i} \cdot \boldsymbol{y'}} \cdot (g^s)^{-p_0(\rho^\star(i))}$.

We claim that if when $Z = e(g_{q+1}, h)$, then the above ciphertext is a valid challenge. The term $C, \hat{C}$ is trivial. For all $x \in \omega'$, we have $f(x) = 0$, hence

$$C'_x = (g^s)^{f'(x)} = (g_q^{f(x)} g^{f'(x)})^s = F_{\mathsf{o}}(x)^s.$$

For $i = 1, \ldots, \ell^\star_{\mathsf{s}}$, we have

$$\begin{aligned} C_i &= (g^\alpha)^{\boldsymbol{M^\star_i} \cdot \boldsymbol{y'}} \prod_{j=1}^{k^\star_{\mathsf{s}}} g^{M^\star_{i,j} s \alpha^j} \cdot (g^s)^{-p_0(\rho^\star(i))} \prod_{j=1}^{k^\star_{\mathsf{s}}} (g^s)^{-M^\star_{i,j}\alpha^j} \\ &= g^{\alpha \boldsymbol{M^\star_i} \cdot \boldsymbol{v}} \cdot (g^s)^{-p(\rho^\star(i))} = g^{\alpha \boldsymbol{M^\star_i} \cdot \boldsymbol{v}} F_{\mathsf{s}}(\rho^\star(i))^{-s}, \end{aligned}$$

which concludes our claim.

Phase 2. $\mathcal{B}$ performs exactly as it did in Phase 1.

Guess. $\mathcal{A}$ outputs $b' \in \{0, 1\}$ for the guess of b. If $b = b'$ then $\mathcal{B}$ outputs 1 (meaning $Z = e(g_{q+1}, h)$). Else, it outputs 0 (meaning Z is random in $\mathbb{G}_T$).

We see that if $(g, h, \boldsymbol{y}_{g,\alpha,q}, Z)$ is sampled from $\mathcal{R}_{BDHE}$ then $\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,\alpha,q}, Z) = 0] = \frac{1}{2}$. On the other hand, if $(g, h, \boldsymbol{y}_{g,\alpha,q}, Z)$ is sampled from $\mathcal{P}_{BDHE}$ then we have $|\Pr[\mathcal{B}(g, h, \boldsymbol{y}_{g,\alpha,q}, Z) = 0] - \frac{1}{2}| \geq \epsilon$. It follows that $\mathcal{B}$ has advantage at least ϵ in solving q-BDHE problem in $\mathbb{G}$. This concludes the proof.

4.3 Some Extended Constructions

We note that an unrestricted scheme where ρ is not necessarily injective, a scheme with CCA security, a scheme based only on Decision Bilinear Diffie-Hellman (DBDH) assumption can be realized similarly to [10]. We can also model $F_{\mathsf{s}}, F_{\mathsf{o}}$ as random oracles and achieve better efficiency and simpler proof as in [10]. In Goyal et al. [6] paper, the KP-ABE for LSSS realizable structures does not have delegation property; while the one for access-tree structures have. We can also base our DP-ABE scheme on the access-tree based KP-ABE. Finally, we can extend the access structures to include non-monotone type ones as in [8].

5 Key Delegation in DP-ABE

We now extend the definition and scheme realizations of DP-ABE to obtain the delegation of keys. We begin with the definition of $\mathsf{Delegate}$ algorithm to be added on.

$\mathsf{Delegate}$: It takes as inputs a private key $\mathsf{sk}_{(\psi,\mathbb{O})}$ of subjective attribute set and objective access structure pair $(\psi, \mathbb{O})$, and another new pair $(\psi', \mathbb{O}')$ intended to derive its key. It outputs the key $\mathsf{sk}_{(\psi',\mathbb{O}')}$ if and only if $\psi' \subseteq \psi$ and $\mathbb{O}' \subseteq \mathbb{O}$.

In other words, key delegation can be realized when the new subjective attribute set is a subset of the original set and the new objective access structure is more restrictive than the original one (or either one condition holds while the other remains the same). In defining this algorithm, we require its correctness that the private key $\mathsf{sk}_{(\psi',\mathbb{O}')}$ output from $\mathsf{Delegate}$ has the same distribution as the one from KeyGen algorithm.

Recall that $\mathcal{U}_\mathsf{s}$ is the universe of subjective attributes and $2^{\mathcal{U}_\mathsf{o}}$ is the full objective access structure. The delegation will start from the master key, which can be considered equivalently as the private key for $(\mathcal{U}_\mathsf{s}, 2^{\mathcal{U}_\mathsf{o}})$. From that, we can consider two types of intermediate states: $(\psi, 2^{\mathcal{U}_\mathsf{o}})$ which can be considered as a key in pure CP-ABE scheme and $(\mathcal{U}_\mathsf{s}, \mathbb{O})$ which can be considered as a key in pure KP-ABE scheme.

Such intermediate keys are indeed already defined generically in any DP-ABE scheme (by instantiating $\mathsf{sk}_{(\psi,\mathbb{O})}$ with $\mathbb{O} = 2^{\mathcal{U}_\mathsf{o}}$ for the first type and $\psi = \mathcal{U}_\mathsf{s}$ for the second type). However, both $2^{\mathcal{U}_\mathsf{o}}$ and $\mathcal{U}_\mathsf{s}$ are of super-polynomial size; therefore, the size of instantiated keys could be very large for any DP-ABE constructions (including our basic DPABE construction). To resolve this, we thus newly define KeyGen for only those two specific types of keys below.

We now describe the delegation scheme for our DPABE scheme as follows. The security proof is postponed to Section A.1.

5.1 Delegating CP-ABE to DP-ABE

$$\boxed{(\mathcal{U}_\mathsf{s}, 2^{\mathcal{U}_\mathsf{o}}) \to (\psi, 2^{\mathcal{U}_\mathsf{o}}) \to (\psi, \mathbb{O})}$$

From the master key $\mathsf{msk} = (\gamma, a)$, it randomly chooses $r \in \mathbb{Z}_p$ and creates a private key for $(\psi, 2^{\mathcal{U}_\mathsf{o}})$ as $\mathsf{sk}_{(\psi,2^{\mathcal{U}_\mathsf{o}})} = (K, \{K_x\}_{x\in\psi}, \hat{K})$ where

$$K = g^r, \quad K_x = F_\mathsf{s}(x)^r, \quad \hat{K} = g^{\gamma+ar}. \tag{5}$$

Note that this is exactly the key in the CP-ABE of Waters [10]. This means that one can seamlessly extend Waters' CP-ABE to ours DP-ABE without having to setup again. The decryption using this key can be done by Eq.(1) but neglecting all the terms related to objective attribute set, ω. Thus, Eq.(1) is simplified to

$$C \cdot \frac{\prod_{i\in I_\mathsf{s}} \left(e(C_i, K) \cdot e(\hat{C}, K_{\rho(i)})\right)^{\mu_i}}{e(\hat{K}, \hat{C})} = \mathcal{M}.$$

From the above private key for $(\psi, 2^{\mathcal{U}_\mathsf{o}})$, we can further delegate to obtain a private key for $(\psi, \mathbb{O})$. Let $\mathbb{O}$ be represented by a LSSS (N, π) as usual. Let N be $\ell_\mathsf{o} \times k_\mathsf{o}$ matrix. The algorithm randomly chooses $r', z_2 \ldots, z_{k_\mathsf{o}}, r_1, \ldots, r_{\ell_\mathsf{o}} \in \mathbb{Z}_p$.

It implicitly lets $\boldsymbol{v} = (\gamma + a(r+r'), z_2, \ldots, z_{k_{\mathsf{o}}})$. It creates the private key $\mathsf{sk}_{(\psi,\mathbb{O})} = (K^{\mathsf{new}}, \{K_x^{\mathsf{new}}\}_{x\in\psi}, \{\hat{K}_i^{\mathsf{new}}, K_i'^{\mathsf{new}}\}_{i=1,\ldots,\ell_{\mathsf{o}}})$ as

$$K^{\mathsf{new}} = K \cdot g^{r'}, \qquad K_x^{\mathsf{new}} = K_x \cdot F_{\mathsf{s}}(x)^{r'},$$
$$\hat{K}_i^{\mathsf{new}} = (\hat{K} \cdot (g^a)^{r'})^{N_{i,1}} g^{\sum_{j=2}^{k_{\mathsf{o}}} N_{i,j} z_j} F_{\mathsf{o}}(\pi(i))^{-r_i}, \qquad K_i'^{\mathsf{new}} = g^{r_i},$$

which distributes exactly the same as in our main scheme; in particular, one can verify that $\hat{K}_i^{\mathsf{new}} = g^{\boldsymbol{N_i}\cdot\boldsymbol{v}} F_{\mathsf{o}}(\pi(i))^{-r_i}$.

5.2 Delegating KP-ABE to DP-ABE

$\boxed{(\mathcal{U}_{\mathsf{s}}, 2^{\mathcal{U}_{\mathsf{o}}}) \to (\mathcal{U}_{\mathsf{s}}, \mathbb{O}) \to (\psi, \mathbb{O})}$

From the master key $\mathsf{msk} = (\gamma, a)$, the algorithm will create a private key for $(\mathcal{U}_{\mathsf{s}}, \mathbb{O})$ as follows. Let $\mathbb{O}$ be represented by a LSSS (N, π) as usual. Let N be $\ell_{\mathsf{o}} \times k_{\mathsf{o}}$ matrix. The algorithm randomly chooses $z_2 \ldots, z_{k_{\mathsf{o}}}, r_1, \ldots, r_{\ell_{\mathsf{o}}} \in \mathbb{Z}_p$. It lets $\boldsymbol{z} = (\gamma, z_2, \ldots, z_{k_{\mathsf{o}}})$. It then creates $\mathsf{sk}_{(\mathcal{U}_{\mathsf{s}},\mathbb{O})} = (\{\hat{K}_i, K_i'\}_{i=1,\ldots,\ell_{\mathsf{o}}})$ where

$$\hat{K}_i = g^{\boldsymbol{N_i}\cdot\boldsymbol{z}} F_{\mathsf{o}}(\pi(i))^{-r_i}, \quad K_i' = g^{r_i}, \tag{6}$$

Note that this is exactly the key in the KP-ABE of Goyal et al. [10]. This means that one can seamlessly extend such KP-ABE schemes to ours DP-ABE without having to setup again. The decryption using this key can be done by Eq.(1) but neglecting all the terms related to subjective attribute set, ψ. Thus, Eq.(1) is simplified to

$$C \cdot \frac{1}{\prod_{j\in I_{\mathsf{o}}} \Big(e(\hat{K}_j, \hat{C}) \cdot e(K_j', C_{\pi(j)}') \Big)^{\nu_j}} = \mathcal{M}.$$

From the above private key for $(\mathcal{U}_{\mathsf{s}}, \mathbb{O})$, we can further delegate to obtain a private key for $(\psi, \mathbb{O})$.The algorithm randomly chooses $r, z_2', \ldots, z_{k_{\mathsf{o}}}', r_1', \ldots, r_{\ell_{\mathsf{o}}}' \in \mathbb{Z}_p$. It creates $\mathsf{sk}_{(\psi,\mathbb{O})} = (K^{\mathsf{new}}, \{K_x^{\mathsf{new}}\}_{x\in\psi}, \{\hat{K}_i^{\mathsf{new}}, K_i'^{\mathsf{new}}\}_{i=1,\ldots,\ell_{\mathsf{o}}})$ as

$$K^{\mathsf{new}} = g^r, \qquad K_x^{\mathsf{new}} = F_{\mathsf{s}}(x)^r,$$
$$\hat{K}_i^{\mathsf{new}} = \hat{K}_i \cdot (g^a)^{N_{i,1} r} g^{\sum_{j=2}^{k_{\mathsf{o}}} N_{i,j} z_j'} F_{\mathsf{o}}(\pi(i))^{-r_i'}, \qquad K_i'^{\mathsf{new}} = K_i' \cdot g^{r_i'},$$

which distributes exactly the same as in our main scheme.

5.3 Delegating in DP-ABE

$\boxed{(\psi, \mathbb{O}) \to (\psi', \mathbb{O}')}$

The delegation from $(\psi, \mathbb{O}) \to (\psi', \mathbb{O})$, where $\psi' \subset \psi$, can be done by deleting the elements K_x where $x \in \psi \setminus \psi'$ and then re-randomizing the other remaining elements in a similar way as delegations stated previously. More precisely, from $\mathsf{sk}_{(\psi,\mathbb{O})} = (K, \{K_x\}_{x\in\psi}, \{\hat{K}_i, K'_i\}_{i=1,\ldots,\ell_{\mathsf{o}}})$, the algorithm creates $\mathsf{sk}_{(\psi',\mathbb{O})} = (K^{\mathsf{new}}, \{K_x^{\mathsf{new}}\}_{x\in\psi'}, \{\hat{K}_i^{\mathsf{new}}, K_i'^{\mathsf{new}}\}_{i=1,\ldots,\ell_{\mathsf{o}}})$ as follows. It first randomly chooses $r', z'_2, \ldots, z'_{k_{\mathsf{o}}}, r'_1, \ldots, r'_{\ell_{\mathsf{o}}} \in \mathbb{Z}_p$ and then computes

$$K^{\mathsf{new}} = K \cdot g^{r'}, \qquad K_x^{\mathsf{new}} = K_x \cdot F_{\mathsf{s}}(x)^{r'},$$

$$\hat{K}_i^{\mathsf{new}} = \hat{K}_i \cdot (g^a)^{N_{i,1} r'} g^{\sum_{j=2}^{k_{\mathsf{o}}} N_{i,j} z'_j} F_{\mathsf{o}}(\pi(i))^{-r'_i}, \qquad K_i'^{\mathsf{new}} = K'_i \cdot g^{r'_i},$$

which distributes exactly the same as a key for $(\psi', \mathbb{O})$.

The delegation from $(\psi, \mathbb{O}) \rightarrow (\psi, \mathbb{O}')$, where $\mathbb{O}'$ is more restrictive than $\mathbb{O}$, can be done on the access-tree based DP-ABE in a similar way to the KP-ABE scheme of Goyal et al. [6], with proper re-randomization.

6 Single-Policy Modes of DP-ABE

In this section, we describe a feature of DP-ABE called encryption in single-policy modes. Suppose that a DP-ABE scheme has been set-up already. The single-policy encryption mode allows an encryptor to still encrypt his message as if it were a KP-ABE or CP-ABE on-the-fly. More specifically, when a message is encrypted in KP-ABE mode with objective attribute set ω, any user with key for $(\psi, \mathbb{O})$ where $\omega \in \mathbb{O}$ can decrypt it regardless of whatever subjective attribute set ψ. Analogously, when a message is encrypted in CP-ABE mode with subjective policy $\mathbb{S}$, any user with key for $(\psi, \mathbb{O})$ where $\psi \in \mathbb{S}$ can decrypt it regardless of whatever objective policy $\mathbb{O}$.

We now describe a simple generic construction and then a more efficient direct construction as follows.

6.1 Generic Construction

As a first attempt, we describe a trivial approach to generically realize encryption in single-policy modes as follows. To encrypt in KP-ABE mode with objective attribute set ω, one just encrypt to $(2^{\mathcal{U}_{\mathsf{s}}}, \omega)$. To encrypt in CP-ABE mode with subjective policy $\mathbb{S}$, one just encrypt to $(\mathbb{S}, \mathcal{U}_{\mathsf{o}})$. However, $2^{\mathcal{U}_{\mathsf{s}}}$ and $\mathcal{U}_{\mathsf{o}}$ are of super-polynomial size; therefore, the size of instantiated ciphertext could be very large for any DP-ABE constructions (including our basic DPABE construction).

To resolve this, we propose a simple generic conversion from any DP-ABE scheme S to a new DP-ABE scheme S' that admits efficient single-policy modes as follows. The idea is to use dummy attributes: one for subjective and one for objective attribute.

$\mathsf{S}'.\mathsf{Setup}$ is exactly the same as $\mathsf{S}.\mathsf{Setup}$ except that it additionally chooses a special subjective attribute $T_{\mathsf{s}} \in \mathcal{U}_{\mathsf{s}}$ and a special objective attribute $T_{\mathsf{o}} \in \mathcal{U}_{\mathsf{o}}$ and adds them into the public key. Both $T_{\mathsf{s}}, T_{\mathsf{o}}$ will not be used as attributes in S'. Next we define

$$\mathsf{S}'.\mathsf{KeyGen}(\mathsf{pk}, \mathsf{msk}, (\psi, \mathbb{O})) = \mathsf{S}.\mathsf{KeyGen}\big(\mathsf{pk}, \mathsf{msk}, (\psi \cup \{T_{\mathsf{s}}\}, \mathbb{O} \cup \{\{T_{\mathsf{o}}\}\})\big).$$

$\mathsf{S}'.\mathsf{Encrypt}$ is done as usual except in the single-policy modes where we define

$$\mathsf{S}'.\mathsf{Encrypt}(\mathsf{pk}, \mathcal{M}, (2^{\mathcal{U}_\mathsf{s}}, \omega)) = \mathsf{S}.\mathsf{Encrypt}\big(\mathsf{pk}, \mathcal{M}, (\{\{T_\mathsf{s}\}\}, \omega)\big),$$
$$\mathsf{S}'.\mathsf{Encrypt}(\mathsf{pk}, \mathcal{M}, (\mathbb{S}, \mathcal{U}_\mathsf{o})) = \mathsf{S}.\mathsf{Encrypt}\big(\mathsf{pk}, \mathcal{M}, (\mathbb{S}, \{T_\mathsf{o}\})\big),$$

which corresponds to KP-ABE and CP-ABE mode respectively. Decryption can be done exactly in the same way as usual.

6.2 Direct Construction

When applying the above generic conversion to our proposed DPABE, the resulting scheme seems to contain some redundancy, in particular, involving using the dummy subjective attribute and the LSSS scheme for the augmented objective access structure $\mathbb{O} \cup \{\{T_\mathsf{o}\}\}$. In this section, we thus also present a direct construction DPABE2 by tweaking the main DPABE construction as follows.

DPABE2.Setup is exactly the same as that of DPABE except that it also includes a special objective attribute $T_\mathsf{o} \in \mathcal{U}_\mathsf{o}$ in the public key. DPABE2.KeyGen is also exactly the same as before except the following. To generate the key $\mathsf{sk}_{(\psi,\mathbb{O})}$, it also includes two new elements $(\hat{K}_{(\mathsf{o})}, K'_{(\mathsf{o})})$ which are computed by first randomly choosing $\tilde{r} \in \mathbb{Z}_p$ and setting

$$\hat{K}_{(\mathsf{o})} = g^{\gamma + ar} F_\mathsf{o}(T_\mathsf{o})^{-\tilde{r}}, \quad K'_{(\mathsf{o})} = g^{\tilde{r}}. \tag{7}$$

Hence the key will be $\mathsf{sk}_{(\psi,\mathbb{O})} = (K, \{K_x\}_{x\in\psi}, \{\hat{K}_i, K'_i\}_{i=1,\ldots,\ell_\mathsf{o}}, \hat{K}_{(\mathsf{o})}, K'_{(\mathsf{o})})$.

For the intermediate states, the key $\mathsf{sk}_{(\psi,2^{\mathcal{U}_\mathsf{o}})}$ is unchanged from Eq.(5), while the key $\mathsf{sk}_{(\mathcal{U}_\mathsf{s},\mathbb{O})}$ is exactly the same as defined in Eq.(6) except that it additionally includes the two above new elements of Eq.(7) albeit setting $r = 0$. The delegation can be done as usual with proper re-randomization.

The encryption DPABE2.Encrypt is exactly the same as usual DPABE except in the single-policy modes which we describe below. To encrypt in KP-ABE mode, *i.e.,* to encrypt to $(2^{\mathcal{U}_\mathsf{s}}, \omega)$, one randomly chooses $s \in \mathbb{Z}_p$ and set the ciphertext to $\mathsf{ct} = (C, \hat{C}, C_0, \{C'_x\}_{x\in\omega})$, where

$$C = \mathcal{M} \cdot (e(g,g)^\gamma)^s, \qquad \hat{C} = g^s,$$
$$C_0 = g^{as}, \qquad C'_x = F_\mathsf{o}(x)^s.$$

The decryption in this case is done by simplifying Eq.(1) to

$$C \cdot \frac{e(C_0, K)}{\prod_{j\in I_\mathsf{o}} \Big(e(\hat{K}_j, \hat{C}) \cdot e(K'_j, C'_{\pi(j)})\Big)^{\nu_j}} = \mathcal{M}.$$

On the other hand, to encrypt in CP-ABE mode, *i.e.,* to encrypt to $(\mathbb{S}, \mathcal{U}_\mathsf{o})$, one just compute as in the usual DPABE.Encrypt but set the ciphertext to $\mathsf{ct} = (C, \hat{C}, \{C_i\}_{i=1,\ldots,\ell_\mathsf{s}}, C')$, where

$$C = \mathcal{M} \cdot (e(g,g)^\gamma)^s, \qquad \hat{C} = g^s,$$
$$C_i = g^{a\lambda_i} F_\mathsf{s}(\rho(i))^{-s}, \qquad C' = F_\mathsf{o}(T_\mathsf{o})^s.$$

The decryption in this case is done by simplifying Eq.(1) to

$$C \cdot \frac{\prod_{i \in I_{\mathsf{s}}} \left(e(C_i, K) \cdot e(\hat{C}, K_{\rho(i)}) \right)^{\mu_i}}{e(\hat{K}_{(\mathsf{o})}, \hat{C}) \cdot e(K'_{(\mathsf{o})}, C')} = \mathcal{M}.$$

The security proof of DPABE2 is given in Section A.2.

7 Conclusions

We presented a new variant of Attribute based encryption (ABE) called Dual-Policy ABE. It is a useful primitive that combines two access control functionalities from Ciphertext-policy ABE and Key-policy ABE. We formalized the notion of Dual-policy ABE and presented an efficient concrete scheme based on an algebraic combination between Goyal et al. KP-ABE [6] and Waters' CP-ABE [10], which are the state-of-the-art schemes for ABE of respective kinds. We further proposed two add-on features: key delegation and single-policy modes of encryption. Key delegation has an interesting property that it also allows the delegation from KP-ABE key of Goyal et al. scheme or CP-ABE key of Waters' scheme to our DP-ABE. Therefore, one can extend those two existing ABE schemes by delegating to DP-ABE seamlessly. Single-policy mode feature allows users to use DP-ABE keys as if it were the vanilla KP-ABE or CP-ABE on-the-fly. This feature allows great flexibility since one DP-ABE key can be used for all three types of ABE (KP,CP,DP ABE).

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-Policy Attribute-Based Encryption. In: IEEE Symposium on Security and Privacy 2007, pp. 321–334 (2007)
2. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. SIAM Journal of Computing 32(3), 586–615 (2003); In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
3. Boneh, D., Hamburg, M.: Generalized Identity Based and Broadcast Encryption Schemes. In: Pieprzyk, J. (ed.) Asiacrypt 2008. LNCS, vol. 5350, pp. 455–470. Springer, Heidelberg (2008)
4. 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)
5. Goyal, V., Jain, A., Pandey, O., Sahai, A.: Bounded ciphertext policy attribute-based encryption. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008 (Track C), Part I. LNCS, vol. 5125, pp. 579–591. Springer, Heidelberg (2008)
6. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM Conference on Computer and Communications Security 2006, pp. 89–98 (2006)
7. 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)

8. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-Based Encryption with Non-Monotonic Access Structures. In: ACM Conference on Computer and Communications Security 2007, pp. 195–203 (2007)
9. Sahai, A., Waters, B.: Fuzzy Identity-Based Encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
10. Waters, B.: Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. Cryptology ePrint archive: report 2008/290

A Security Proofs of Schemes with Extended Features

A.1 Security Proof of the Scheme with Delegation

In this section, we describe the security proof of the scheme with delegation given in Section 5. The only difference from our basic construction in Section 4.1 is that we newly re-define the private key $\mathsf{sk}_{(\psi,2^{\mathcal{U}_\circ})}, \mathsf{sk}_{(\mathcal{U}_\mathsf{s},\mathbb{O})}$, for the intermediate states. According to the security definition, the adversary can also query for the key $\mathsf{sk}_{(\mathcal{U}_\mathsf{s},\mathbb{O})}$ if $\omega^\star$ does not satisfy $\mathbb{O}$ and the key $\mathsf{sk}_{(\psi,2^{\mathcal{U}_\circ})}$ if ψ does not satisfy $\mathbb{S}^\star$. Here we recall that $(\mathbb{S}^\star, \omega^\star)$ is the target subjective access structure and objective attribute set pair. Therefore, it suffices to show how to simulate these two types of keys in Phase 1 (and 2), in addition to the proof of the main scheme (*cf.* Section 4.2).

For the first type, the simulator $\mathcal{B}$ answers the query for $\mathsf{sk}_{(\mathcal{U}_\mathsf{s},\mathbb{O})}$ such that $\omega^\star$ does not satisfy $\mathbb{O}$ by simulating the private key elements in exactly the same way as in the Case 1 in Phase 1 in the proof of the main scheme, albeit setting $r = 0$ and neglecting the term K, K_x. The resulting simulated key $(\{\hat{K}_i, K'_i\}_{i=1,\ldots,\ell_\circ})$ is distributed as the key $\mathsf{sk}_{(\mathcal{U}_\mathsf{s},\mathbb{O})}$ in the real scheme (*cf.* Eq.(6)). This holds thanks to the correctness of simulation for $\mathsf{sk}_{(\psi,\mathbb{O})}$ in the proof of our main scheme and the fact that $\mathsf{sk}_{(\mathcal{U}_\mathsf{s},\mathbb{O})}$ as defined in Eq.(6) simplifies $\mathsf{sk}_{(\psi,\mathbb{O})}$ as defined in the main scheme with r being set to $r = 0$.

For the second type, the simulator $\mathcal{B}$ answers the query for $\mathsf{sk}_{(\psi,2^{\mathcal{U}_\circ})}$ such that ψ does not satisfy $\mathbb{S}^\star$ as follows. Since the elements $(K, \{K_x\}_{x\in\psi})$ in both the key $\mathsf{sk}_{(\psi,2^{\mathcal{U}_\circ})}$ defined in Eq.(5) and the key $\mathsf{sk}_{(\psi,\mathbb{O})}$ of the main scheme are the same, we just simulate $(K, \{K_x\}_{x\in\psi})$ exactly as in the Case 2 in Phase 1 in the proof of the main scheme. It then computes $\hat{K}$ as $\hat{K} = g^{\gamma'} g_1^{r'} \prod_{k=2}^{k_\mathsf{s}^\star+m} (g_{q-k+2})^{w_k}$,which can be verified that $\hat{K} = g^{\gamma+\alpha r}$ as required (recall that in the simulation, r is implicitly defined in Eq.(4) and $a = \alpha$).

Remark 2. In the security proof of the main scheme in Section 4.2, we could have done a simpler simulation if the key delegation were already defined there. For Case 1, it suffices to compute the key $\mathsf{sk}_{(\mathcal{U}_\mathsf{s},\mathbb{O})}$ and then delegate to $\mathsf{sk}_{(\psi,\mathbb{O})}$ to answer the query. For Case 2, it suffices to compute the key $\mathsf{sk}_{(\psi,2^{\mathcal{U}_\circ})}$ and then delegate to $\mathsf{sk}_{(\psi,\mathbb{O})}$. However, we believe that separating the key delegation feature from the basic scheme makes its description easier to follow.

A.2 Security Proof of the Scheme with Single-Policy Modes

In this section, we give a sketch of the security proof for this tweaked scheme $\mathsf{DPABE2}$ given in Section 6.2. Note that the only differences from the main proof are as follows. First we must also consider two new possible target pair types of $(2^{\mathcal{U}_\mathsf{s}}, \omega)$ and $(\mathbb{S}, \mathcal{U}_\mathsf{o})$ for the challenge ciphertext. Second, we must also simulate the two new private key elements for each query.

We first consider the normal case where the adversary announces the target pair of type $(\mathbb{S}^\star, \omega^\star)$ in the Init phase. In this case, the proof follows exactly the main proof except that the simulator also simulates additional key components. For Case 1 of Phase 1 in the main proof, it computes the additional keys as

$$\hat{K}_{(\mathsf{o})} = g^{\gamma'} g^{\alpha r} g_1^{-f'(T_\mathsf{o})/f(T_\mathsf{o})} F_\mathsf{o}(T_\mathsf{o})^{-\tilde{r}'}, \quad K'_{(\mathsf{o})} = g^{\tilde{r}'} g_1^{1/f(T_\mathsf{o})}, \tag{8}$$

where it randomly chooses $\tilde{r}' \in \mathbb{Z}_p$. It can be verified that this distributes as in Eq.(7) with $\tilde{r} = \tilde{r}' + 1/f(T_\mathsf{o})$. For Case 2, the simulator can compute $g^{\gamma + \alpha r}$ and thus can generate the elements of Eq.(7) above.

Next, we consider the case where the adversary announces the target pair of type $(2^{\mathcal{U}_\mathsf{s}}, \omega^\star)$ in the Init phase, *i.e.,* the challenge ciphertext will be in KP-ABE mode. In Setup phase, the simulator chooses $a \in \mathbb{Z}_p$ and $h_0, \ldots, h_{m'} \in \mathbb{G}$ randomly (in particular, instead of setting $a = \alpha$ as previously done). The remaining elements of the public key are simulated as in the main proof. In Phase 1, it suffices to simulate the key for $(\mathcal{U}_\mathsf{s}, \mathbb{O})$ such that $\omega^\star$ does not satisfy $\mathbb{O}$. This can be done in exactly the same way as before (*cf.* Section A.1, first type), albeit it also includes two new elements as in Eq.(8) with $r = 0$. In Challenge phase, the term $C, \hat{C}, C'_x$ can be simulated as usual. In addition, it just sets $C_0 = \hat{C}^a$. The rest follows from the main proof.

Finally, we consider the case where the adversary announces the target pair of type $(\mathbb{S}^\star, \mathcal{U}_\mathsf{o})$ in the Init phase, *i.e.,* the challenge ciphertext will be in CP-ABE mode. In this case, the proof follows exactly the main proof that is instantiated with the selective target pair $(\mathbb{S}^\star, \{T_\mathsf{o}\})$. Note also that it suffices to simulate the key for $\mathsf{sk}_{(\psi, 2^{\mathcal{U}_\mathsf{o}})}$ such that ψ does not satisfy $\mathbb{S}^\star$. Such a key does not include the two new elements of Eq.(7).

Construction of Threshold Public-Key Encryptions through Tag-Based Encryptions

Seiko Arita and Koji Tsurudome

Institute of Information Security,
Yokohama, Kanagawa, Japan
{arita,mgs068101}@iisec.ac.jp

Abstract. In this paper, we propose a notion of threshold tag-based encryption schemes that simplifies the notion of threshold identity-based encryption schemes, and we show a conversion from any stag-CCA-secure threshold tag-based encryption schemes to CCA-secure threshold public-key encryption schemes. Moreover, we give two concrete constructions of stag-CCA-secure threshold tag-based encryption schemes, under the decisional bilinear Diffie-Hellman assumption and the decisional linear assumption, respectively. Thus, we obtain two concrete constructions of threshold public-key encryption schemes, both of which are non-interactive, robust and can be proved secure without random oracle model. Our threshold public-key encryption schemes are conceptually more simple and shown to be more efficient than those of Boneh, Boyen and Halevi.

Keywords: threshold public-key encryption schemes, tag-based encryption schemes, the decisional bilinear Diffie-Hellman assumption, the decisional linear assumption.

1 Introduction

A threshold public-key encryption scheme is a public-key encryption scheme where a private key is distributed and shared among several decryption servers and some number of those decryption servers must cooperate to decrypt any ciphertext [2,4,9]. In a model of k-out-of-n threshold public-key encryption scheme, an entity, called *combiner*, has a ciphertext C that it wishes to decrypt. The combiner sends C to the decryption servers, and receives partial decryption shares from at least k out of the n decryption servers. It then combines these k partial decryptions into a complete decryption of C. Ideally, it is desirable that there is no other interaction in the system, namely the servers need not talk to each other during decryption. Such threshold systems are called non-interactive. Often one requires that threshold decryption be robust, namely if threshold decryption of a valid ciphertext fails, the combiner can identify the decryption servers that supplied invalid partial decryptions.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 186–200, 2009.

In 2006, Boneh, Boyen and Halevi [2] gave a construction of CCA-secure threshold public-key encryption scheme by converting a sID-CPA-secure threshold *identity-based* encryption scheme into it, which in turn is constructed based on the decisional bilinear Diffie-Hellman assumption. Their CCA-secure threshold public-key encryption scheme is the first one that is non-interactive, robust and can be proved secure without random oracle model.

On the other hand, in 2006, Kiltz [6] proposed a notion of tag-based encryption scheme through simplifying the notion of identity-based encryption schemes [3,7], and gave a transformation from any stag-CCA-secure tag-based encryption schemes to CCA-secure public-key encryption schemes.

In this paper, we propose a notion of threshold tag-based encryption schemes that simplifies the notion of threshold identity-based encryption schemes in a similar way as [6], and then we show a conversion from any stag-CCA-secure threshold tag-based encryption schemes to CCA-secure threshold public-key encryption schemes, that is an adaption of the CHK transform [5] to the setting of threshold encryption. Moreover, we give two concrete constructions of stag-CCA-secure threshold tag-based encryption schemes, that are non-interactive, robust and can be proved without random oracle model, under the decisional bilinear Diffie-Hellman assumption and the decisional linear assumption, respectively. Thus, we obtain two concrete constructions of threshold public-key encryption schemes, through applying the conversion to the two threshold tag-based encryption schemes, both of which are non-interactive, robust and can be proved secure without random oracle model.

In the threshold identity-based encryption scheme of [2], a decryption share is regarded as a ciphertext of private key share corresponding to decrypter's ID. But in our threshold tag-based encryption scheme, a decryption share can be regarded naturally as a partial decrypted ciphertext. As a result, our threshold public-key encryption schemes, obtained through the conversion, are conceptually more simple and shown to be more efficient than those of [2].

2 Threshold Tag-Based Encryptions and Their Conversion to Threshold Public-Key Encryptions

In this section, after reviewing the definition of threshold public-key encryptions and their security following [2], we propose a notion of threshold tag-based encryptions and show a conversion from any stag-CCA-secure threshold tag-based encryption schemes to CCA-secure threshold public-key encryption schemes.

2.1 Threshold Public-Key Encryption

Scheme. A threshold public-key encryption scheme TPKE consists of five algorithms:

$$\mathsf{TPKE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{ShareDec}, \mathsf{ShareVf}, \mathsf{Combine}).$$

Setup takes as input the number of decryption servers n, a threshold k $(1 \leq k \leq n)$, and a security parameter Λ. It outputs a triple (PK, VK, SK) where PK

is a public key, VK is a verification key and $SK = (SK_1, \cdots, SK_n)$ is a vector of n private key shares. $\mathsf{Encrypt}$ takes as input a public key PK and a message M, and it outputs a ciphertext C. $\mathsf{ShareDec}$ takes as input a public key PK, a ciphertext C, and the i-th private key share (i, SK_i). It outputs a decryption share μ_i of the encrypted message, or a special symbol $(i, \perp)$. $\mathsf{ShareVf}$ takes as input a public key PK, verification key VK, ciphertext C and a decryption share μ_i. It outputs valid or $\mathsf{invalid}$. When the output is valid, we say that μ_i is a valid decryption share of C. $\mathsf{Combine}$ takes as input PK, VK, ciphertext C, and k decryption shares $\{\mu_1, \cdots, \mu_k\}$. It outputs a message M or $\perp$.

For any output (PK, VK, SK) of $\mathsf{Setup}(n, k, \Lambda)$, we require the two consistency properties:

1. For any valid ciphertext C, if $\mu_i \leftarrow \mathsf{ShareDec}(PK, i, SK_i, C)$, then $\mathsf{ShareVf}(PK, VK, C, \mu_i)$ is valid.
2. If C is the output of $\mathsf{Encrypt}(PK, M)$ and $S = \{\mu_1, \cdots, \mu_k\}$ is a set of decryption shares $\mu_i \leftarrow \mathsf{ShareDec}(PK, i, SK_i, C)$ for k distinct private key shares in SK, then $\mathsf{Combine}(PK, VK, C, S) = M$.

Security. Security of threshold public-key encryption scheme TPKE is defined in terms of chosen ciphertext security and decryption consistency. *Chosen ciphertext security* is defined using the following game between a challenger and an adversary. Both are given n, k, Λ as input.

1. Init. The adversary outputs a set $S \subset \{1, \cdots, n\}$ of $k-1$ decryption servers to corrupt.
2. Setup. The challenger runs $\mathsf{Setup}(n, k, \Lambda)$ to obtain a random instance (PK, VK, SK). It gives the adversary PK, VK, and all (j, SK_j) for $j \in S$.
3. Query phase 1. The adversary adaptively issues decryption queries (C, i) where $C \in \{0,1\}^*$ and $i \in \{1, \cdots, n\}$. The challenger responds with $\mathsf{ShareDec}(PK, i, SK_i, C)$.
4. Challenge. The adversary outputs two messages M_0, M_1 of equal length. The challenger picks a random $b \in \{0,1\}$ and lets $C^* \leftarrow \mathsf{Encrypt}(PK, M_b)$. It gives C^* to the adversary.
5. Query phase 2. The adversary issues further decryption queries (C, i), under the constraint that $C \neq C^*$. The challenger responds as in Query Phase 1.
6. Guess. The adversary outputs its guess $b' \in \{0,1\}$ for b and wins the game if $b = b'$.

We define an advantage of adversary $\mathcal{A}$ for threshold public-key encryption scheme TPKE with respect to chosen ciphertext security as $\mathsf{Adv}^{cca}_{\mathcal{A},\mathsf{TPKE},n,k}(\Lambda) = |\Pr[b = b'] - 1/2|$.

Decryption consistency is defined using the following game. The game starts with the Init, Setup, and Query phase 1 steps as in the game above. The adversary then outputs a ciphertext C and two sets of decryption shares $S = \{\mu_1, \cdots, \mu_k\}$ and $S' = \{\mu'_1, \cdots, \mu'_k\}$ each of size k. The adversary wins if:

- The shares in S and S' are valid decryption shares for C under VK.
- S and S' each contain decryption shares from k distinct servers.
- $\mathsf{Combine}(PK, VK, C, S) \neq \mathsf{Combine}(PK, VK, C, S')$, with either side not equal to $\perp$.

We let $\mathsf{Adv}^{dc}_{\mathcal{A},\mathsf{TPKE},n,k}(\Lambda)$ denote the probability that the adversary $\mathcal{A}$ wins this game.

Definition 1. *We say that a threshold public-key encryption scheme* TPKE *is* CCA-secure *if for any n, k $(1 \leq k \leq n)$ and any probabilistic polynomial time algorithm $\mathcal{A}$, both of the functions $\mathsf{Adv}^{cca}_{\mathcal{A},\mathsf{TPKE},n,k}(\Lambda)$ and $\mathsf{Adv}^{dc}_{\mathcal{A},\mathsf{TPKE},n,k}(\Lambda)$ are negligible.*

2.2 Threshold Tag-Based Encryption

A notion of threshold tag-based encryptions is obtained by simplifying threshold identity-based encryption schemes in a similar way as [6] in the non-threshold setting. Threshold tag-based encryptions simply needs a *tag* as input in addition to ordinary inputs of threshold encryptions.

Scheme. A threshold tag-based encryption scheme TTBE consists of five algorithms:

$$\mathsf{TTBE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{ShareDec}, \mathsf{ShareVf}, \mathsf{Combine}).$$

Setup takes as input the number of decryption servers n, a threshold k $(1 \leq k \leq n)$ and a security parameter Λ. It outputs a triple (PK, VK, SK) where PK is a public key, VK is a verification key, and $SK = (SK_1, \cdots, SK_n)$ is a vector of n private key shares. Encrypt takes as input a public key PK, a *tag* t and a message M, and it outputs a ciphertext C. ShareDec takes as input a public key PK, a ciphertext C, a *tag* t, and a i-th private key share (i, SK_i). It outputs a decryption share μ_i of the encrypted message, or a special symbol $(i, \perp)$. ShareVf takes as input PK, VK, a ciphertext C, a *tag* t and a decryption share μ_i. It outputs valid or invalid. When the output is valid, we say that μ_i is a valid decryption share of C. Combine takes as input PK, VK, a ciphertext C, a *tag* t and k decryption shares $\{\mu_1, \cdots, \mu_k\}$. It outputs a message M or $\perp$.

As in the threshold public-key encryption scheme, we require the following two consistency properties. Let (PK, VK, SK) be the output of $\mathsf{Setup}(n, k, \Lambda)$.

1. For any tuple (C, t) of a valid ciphertext and a tag, if $\mu_i \leftarrow \mathsf{ShareDec}$ (PK, i, SK_i, C, t), then $\mathsf{ShareVf}(PK, VK, C, t, \mu_i) = \mathsf{valid}$.
2. If C is the output of $\mathsf{Encrypt}(PK, t, M)$ and $S = \{\mu_1, \cdots, \mu_k\}$ is a set of decryption shares $\mu_i \leftarrow \mathsf{ShareDec}(PK, i, SK_i, C, t)$ for k distinct private key shares in SK, then $\mathsf{Combine}(PK, VK, C, t, S) = M$.

Security. Security of threshold tag-based encryption scheme TTBE is defined in terms of stag-chosen-ciphertext security and stag decryption consistency. *Stag chosen ciphertext security* is defined using the following game between a challenger and an adversary. Both are given n, k, Λ as input.

1. Init. The adversary outputs a target tag t^* that it wants to attack and a set of $k-1$ decryption servers $S \subset \{1, \cdots, n\}$ that it wants to corrupt.
2. Setup. The challenger runs $\mathsf{Setup}(n, k, \Lambda)$ to obtain a random instance (PK, VK, SK). It gives the adversary PK, VK, and all (j, SK_j) for $j \in S$.
3. Query phase 1. The adversary adaptively issues decryption share queries $((C, t), i)$ with $i \in \{1, \cdots, n\}$, under the constraint that $t \neq t^*$. The challenger responds with $\mathsf{ShareDec}(PK, i, SK_i, C, t)$.
4. Challenge. The adversary outputs two messages M_0, M_1 of equal length. The challenger picks a random $b \in \{0, 1\}$ and lets $C^* \leftarrow \mathsf{Encrypt}(PK, t^*, M_b)$. It gives C^* to the adversary.
5. Query phase 2. The adversary adaptively issues decryption share queries $((C, t), i)$ with $i \in \{1, \cdots, n\}$, under the constraint that $t \neq t^*$. The challenger responds as in phase 1.
6. Guess. The adversary outputs its guess $b' \in \{0, 1\}$ for b and wins the game if $b = b'$.

We define an advantage of adversary $\mathcal{A}$ for threshold tag-based encryption scheme TTBE with respect to stag-chosen-ciphertext security as $\mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE},n,k}(\Lambda)$ $= |\Pr[b = b'] - 1/2|$.

Stag decryption consistency is defined using the following game. The game starts with the Init, Setup and Query phase 1 steps as in the game above. The adversary then outputs a tag t, a ciphertext C and two sets of decryption shares $S = \{\mu_1, \cdots, \mu_k\}$ and $S' = \{\mu'_1, \cdots, \mu'_k\}$ each of size k. The adversary wins if:

1. The shares in S and S' are valid decryption shares for (C, t) under VK.
2. S and S' each contain decryption shares from k distinct servers.
3. $\mathsf{Combine}(PK, VK, C, t, S) \neq \mathsf{Combine}(PK, VK, C, t, S')$, with either side not equal to $\perp$.

We let $\mathsf{Adv}^{stag-dc}_{\mathcal{A},\mathsf{TTBE},n,k}(\Lambda)$ denote the probability that the adversary $\mathcal{A}$ wins this game.

Definition 2. *We say that a threshold tag-based encryption scheme* TTBE *is* stag-CCA-secure *if for any n, k $(1 \leq k \leq n)$ and any probabilistic polynomial time algorithm $\mathcal{A}$, both of the functions* $\mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE},n,k}(\Lambda)$ *and* $\mathsf{Adv}^{stag-dc}_{\mathcal{A},\mathsf{TTBE},n,k}(\Lambda)$ *are negligible.*

2.3 Conversion from Threshold Tag-Based Encryption Schemes into Threshold Public-Key Encryption Schemes

In this section we show a conversion from any stag-CCA-secure threshold tag-based encryption scheme to CCA-secure threshold public-key encryption scheme. The conversion is a direct adjustment of the conversions of [5,6] into the threshold setting.

We convert a given threshold tag-based encryption scheme

$$\mathsf{TTBE} = (\mathsf{Setup_{ttbe}}, \mathsf{Encrypt_{ttbe}}, \mathsf{ShareDec_{ttbe}}, \mathsf{ShareVf_{ttbe}}, \mathsf{Combine_{ttbe}})$$

into a threshold public-key encryption scheme

$$\mathsf{TPKE} = \mathrm{TT2TP}(\mathsf{TTBE}, \mathsf{S}) = (\mathsf{Setup_{tpke}}, \mathsf{Encrypt_{tpke}}, \mathsf{ShareDec_{tpke}}, \mathsf{ShareVf_{tpke}}, \mathsf{Combine_{tpke}})$$

using a strong one-time signature $\mathsf{S} = (\mathsf{KG}, \mathsf{SGN}, \mathsf{VF})$ as in Figure 1.

```
$\textbf{Setup}_{\textbf{tpke}}(n, k, \Lambda)$ :
  $(PK, VK, SK) \leftarrow \mathsf{Setup_{ttbe}}(n, k, \Lambda)$, output $(PK, VK, SK)$.
$\textbf{Encrypt}_{\textbf{tpke}}(PK, M)$ :
  $(sigk, verk) \leftarrow \mathsf{KG}(\Lambda)$, $C_{ttbe} \leftarrow \mathsf{Encrypt_{ttbe}}(PK, verk, M)$, $\sigma \leftarrow \mathsf{SGN}(sigk, C_{ttbe})$;
  output $C_{tpke} = (C_{ttbe}, verk, \sigma)$.
$\textbf{ShareDec}_{\textbf{tpke}}(PK, i, SK_i, C_{tpke} = (C_{ttbe}, verk, \sigma))$ :
  If $\mathsf{VF}(verk, C_{ttbe}, \sigma) = \mathsf{invalid}$ then output $\mu_i = (i, \perp)$,
   else output $\mathsf{ShareDec_{ttbe}}(PK, i, SK_i, C_{ttbe}, verk)$.
$\textbf{ShareVf}_{\textbf{tpke}}(PK, VK, C_{tpke} = (C_{ttbe}, verk, \sigma), \mu_i)$ :
  If $\mathsf{VF}(verk, C_{ttbe}, \sigma) = \mathsf{invalid}$ then output $\mathsf{invalid}$,
   else output $\mathsf{ShareVf_{ttbe}}(PK, VK, C_{ttbe}, verk, \mu_i)$.
$\textbf{Combine}_{\textbf{tpke}}(PK, VK, C_{tpke} = (C_{ttbe}, verk, \sigma), \{\mu_1, \cdots, \mu_k\})$ :
  If $\exists i$, $\mu_i = (i, \perp)$ or $\mathsf{ShareVf_{tpke}}(PK, VK, C_{tpke}, \mu_i) = \mathsf{invalid}$ then output $\perp$,
   else output $\mathsf{Combine_{ttbe}}(PK, VK, C_{ttbe}, verk, \{\mu_1, \cdots, \mu_k\})$.
```

Fig. 1. TT2TP: Conversion from threshold tag-based encryption schemes to threshold public-key encryption schemes

Theorem 1. *If a threshold tag-based encryption scheme* TTBE *is stag-CCA-secure and* S *is a strong one-time signature, then the threshold public-key encryption scheme* $\mathsf{TPKE} = \mathrm{TT2TP}(\mathsf{TTBE}, \mathsf{S})$ *is CCA-secure.*

More precisely, for an arbitrary efficient adversary $\mathcal{A}$ *against chosen ciphertext security of* TPKE*, there exists an efficient algorithm* $\mathcal{B}$ *against stag-chosen-ciphertext security of the underlying* TTBE *and a forger* $\mathcal{F}$ *of the underlying* S *that satisfy*

$$\mathsf{Adv}^{cca}_{\mathcal{A},\mathsf{TPKE},n,k}(\Lambda) \leq \mathsf{Adv}^{stag-cca}_{\mathcal{B},\mathsf{TTBE},n,k}(\Lambda) + \mathsf{Adv}^{ot-cma}_{\mathcal{F},\mathsf{S}}(\Lambda).$$

(Here, $\mathsf{Adv}^{ot-cma}_{\mathcal{F},\mathsf{S}}$ *denotes the advantage of forger* $\mathcal{F}$ *against one-time signature* S *in the usual game of strong chosen-message attack with at most one signing query.) Similarly, for an arbitrary efficient adversary* $\mathcal{A}'$ *against decryption consistency of* TPKE*, there exists an efficient algorithm* $\mathcal{B}'$ *against stag decryption consistency of the underlying* TTBE *and a forger* $\mathcal{F}'$ *of the underlying* S *that satisfy*

$$\mathsf{Adv}^{dc}_{\mathcal{A}',\mathsf{TPKE},n,k}(\Lambda) \leq \mathsf{Adv}^{stag-dc}_{\mathcal{B}',\mathsf{TTBE},n,k}(\Lambda) + \mathsf{Adv}^{ot-cma}_{\mathcal{F}',\mathsf{S}}(\Lambda).$$

Proof. First, we consider chosen ciphertext security of TPKE. Let $\mathcal{A}$ be an arbitrary efficient adversary against chosen ciphertext security of TPKE. Using adversary $\mathcal{A}$, we build an algorithm $\mathcal{B}$ that attacks stag-chosen-ciphertext security of the underlying TTBE.

Algorithm $\mathcal{B}$ proceeds as follows:

1. Initialization. Given input $(n, k, \varLambda)$ algorithm $\mathcal{B}$ runs $\mathcal{A}$ on the same input to obtain a list S $(\subset \{1, \cdots, n\})$ of the $k-1$ servers that $\mathcal{A}$ wishes to corrupt. Next, $\mathcal{B}$ runs KG on $\varLambda$ to obtain a signing key $sigk^*$ and a verification key $verk^*$. It outputs the set S and the target tag $t^* = verk^*$ to the TTBE challenger.
2. Setup. The TTBE challenger runs $\mathsf{Setup}_{\mathsf{ttbe}}(n, k, \varLambda)$ to obtain (PK, VK, SK). It gives $\mathcal{B}$ the values PK, VK, and all (j, SK_j) for $j \in S$. Algorithm $\mathcal{B}$ forwards these values to $\mathcal{A}$.
3. Query Phase 1. Adversary $\mathcal{A}$ adaptively issues decryption queries of the form (C_{tpke}, i) where $C_{tpke} = (C_{ttbe}, verk, \sigma)$ and $i \in \{1, \cdots, n\}$. For each such a query (C_{tpke}, i), $\mathcal{B}$ proceeds as follows:
   (a) If $\mathsf{VF}(verk, C_{ttbe}, \sigma) = \mathsf{invalid}$ then $\mathcal{B}$ gives $\mu_i = (i, \perp)$ to $\mathcal{A}$.
   (b) Else if $verk = t^*$ then $\mathcal{B}$ outputs $b \overset{\$}{\leftarrow} \{0, 1\}$ and aborts.
   (c) Else $\mathcal{B}$ issues a decryption query $((C_{ttbe}, verk), i)$ to own TTBE decryption oracle and obtains a decryption share μ_i in return. It gives the decryption share μ_i to $\mathcal{A}$.
4. Challenge. Adversary $\mathcal{A}$ outputs two equal-length messages M_0 and M_1. $\mathcal{B}$ forwards these M_0 and M_1 to its own TTBE challenger. The TTBE challenger responds with the encryption C^*_{ttbe} of M_b under t^* for some $b \in \{0, 1\}$. $\mathcal{B}$ then runs SGN on $(sigk^*, C^*_{ttbe})$ to obtain a signature σ^*, and it gives $C^*_{tpke} = (C^*_{ttbe}, t^*, \sigma^*)$ to $\mathcal{A}$ as challenge ciphertext.
5. Query Phase 2. $\mathcal{A}$ continues to issue decryption queries $(C_{tpke}\ (\neq C^*_{tpke}), i)$. $\mathcal{B}$ responds as in Query Phase 1.
6. Guess. Eventually, $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$ for b. $\mathcal{B}$ forwards b' to the TTBE challenger and wins the game if $b = b'$.

This completes the description of algorithm $\mathcal{B}$.

Let Abort be the event that $\mathcal{B}$ aborts in Query Phase 1 or 2 during the simulation. As easily seen, as long as Abort does not happen, $\mathcal{B}$'s simulation of TPKE challenger is perfect. Therefore, we have $|\mathsf{Adv}^{stag-cca}_{\mathcal{B},\mathsf{TTBE},n,k}(\varLambda) - \mathsf{Adv}^{cca}_{\mathcal{A},\mathsf{TPKE},n,k}(\varLambda)| <$ Pr[Abort]. By definition, Abort means $\mathcal{A}$'s forgery of valid signature σ under verification key $verk^*$, and it leads to a forger $\mathcal{F}$ of S satisfying Pr[Abort] $\leq \mathsf{Adv}^{ot-cma}_{\mathcal{F},\mathsf{S}}$. Thus, $\mathsf{Adv}^{cca}_{\mathcal{A},\mathsf{TPKE},n,k}(\varLambda) \leq \mathsf{Adv}^{stag-cca}_{\mathcal{B},\mathsf{TTBE},n,k}(\varLambda) + \mathsf{Adv}^{ot-cma}_{\mathcal{F},\mathsf{S}}(\varLambda)$.

Second, we see decryption consistency of TPKE. Let $\mathcal{A}'$ be an arbitrary efficient adversary against decryption consistency of TPKE. Using adversary $\mathcal{A}'$, we build an algorithm $\mathcal{B}'$ that attacks stag decryption consistency of the underlying TTBE.

Algorithm $\mathcal{B}'$ proceeds exactly as algorithm $\mathcal{B}$, until $\mathcal{A}'$ outputs the challenge $(\hat{C_{tpke}} = (\hat{C_{ttbe}}, \widehat{verk}, \hat{\sigma}), S, S')$, and then $\mathcal{B}'$ outputs $(\widehat{verk}, \hat{C_{ttbe}}, S, S')$ after verifying validity of $\hat{\sigma}$ under $\widehat{verk}$.

Just as in the case of chosen ciphertext security, let Abort be the event that $\mathcal{B}'$ aborts in Query Phase 1 during the simulation. Then, as above, $|\mathsf{Adv}^{stag-dc}_{\mathcal{B}',\mathsf{TPKE},n,k}(\varLambda)$ $- \mathsf{Adv}^{dc}_{\mathcal{A}',\mathsf{TTBE},n,k}(\varLambda)| <$ Pr[Abort]. Again, Abort leads to a forger $\mathcal{F}'$ of S, and we have $\mathsf{Adv}^{dc}_{\mathcal{A}',\mathsf{TPKE},n,k}(\varLambda) \leq \mathsf{Adv}^{stag-dc}_{\mathcal{B}',\mathsf{TTBE},n,k}(\varLambda) + \mathsf{Adv}^{ot-cma}_{\mathcal{F}',\mathsf{S}}(\varLambda)$. □

3 Construction of Threshold Tag-Based Encryption Schemes

In this section, we construct two concrete stag-CCA-secure threshold tag-based encryption schemes based on the decisional bilinear Diffie-Hellman assumption and on the decisional linear assumption, respectively.

3.1 Preliminaries

We recall necessary primitives around bilinear maps.

Bilinear Maps. Let $\mathbb{G}$ be a group of prime order p with generator g. Let $\mathbb{G}_1$ be another group of prime order p. A *bilinear map* $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ is a map with the properties:

1. For all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, it holds $e(u^a, v^b) = e(u, v)^{ab}$.
2. $e(g, g) \neq 1$.
3. For all u, v, $e(u, v)$ is efficiently computable.

Decisional Bilinear Diffie-Hellman Assumption. If a bilinear Diffie-Hellman tuple $(g, g^a, g^b, g^c, e(g, g)^{abc})$ is indistinguishable from a bilinear random tuple $(g, g^a, g^b, g^c, e(g, g)^d)$, we say the decisional bilinear Diffie-Hellman assumption holds. More formally, as for algorithm G_{DBDH} that takes a security parameter Λ and outputs order p, generator g, and descriptions of groups $\mathbb{G}$ and $\mathbb{G}_1$ with bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$, the following two experiments are defined. $\mathbf{Exp}^{\mathsf{bdh-1}}_{G_{DBDH},\mathcal{A}}$ on input Λ generates $param = (p, g, \mathbb{G}, \mathbb{G}_1, e)$ by $G_{DBDH}(\Lambda)$ and chooses three random elements a, b, c from $\mathbb{Z}_p$. Then it invokes $\mathcal{A}$ on $(param, g, g^a, g^b, g^c, e(g, g)^{abc})$ and returns its output. On a while, $\mathbf{Exp}^{\mathsf{bdh-2}}_{G_{DBDH},\mathcal{A}}$ chooses four random elements a, b, c, d from $\mathbb{Z}_p$ and returns $\mathcal{A}(param, g, g^a, g^b, g^c, e(g, g)^d)$.

We say that the *decisional bilinear Diffie-Hellman (DBDH) assumption* holds for G_{DBDH} if for any probabilistic polynomial time algorithm $\mathcal{A}$, $\mathsf{Adv}^{dbdh}_{\mathcal{A},G_{DBDH}}(\Lambda) \overset{def}{=} \left|\Pr[\mathbf{Exp}^{\mathsf{bdh-1}}_{G_{DBDH},\mathcal{A}}(\Lambda) = 1] - \Pr[\mathbf{Exp}^{\mathsf{bdh-2}}_{G_{DBDH},\mathcal{A}}(\Lambda) = 1]\right|$ is a negligible function of Λ.

Decisional Linear Assumption. If a linear tuple $(g_1, g_2, z, g_1^{r_1}, g_2^{r_2}, z^{r_1+r_2})$ is indistinguishable from a random tuple $(g_1, g_2, z, g_1^{r_1}, g_2^{r_2}, z^s)$, we say the decisional linear assumption holds. More formally, as for algorithm G_{DLIN} that takes a security parameter Λ and outputs order p, generator g, and descriptions of groups $\mathbb{G}$ and $\mathbb{G}_1$ with bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$, the following two experiments are defined. $\mathbf{Exp}^{\mathsf{lin-1}}_{G_{DLIN},\mathcal{A}}$ on input Λ generates $param = (p, g, \mathbb{G}, \mathbb{G}_1, e)$ by $\mathcal{G}_{DLIN}(\Lambda)$ and chooses four random elements u, v, r_1, r_2 from $\mathbb{Z}_p$. Then it invokes $\mathcal{A}$ on $(param, g, g^u, g^v, g^{r_1}, g^{ur_2}, g^{v(r_1+r_2)})$ and returns its output. On a while, $\mathbf{Exp}^{\mathsf{lin-2}}_{G_{DLIN},\mathcal{A}}$ chooses five random elements u, v, r_1, r_2, s from $\mathbb{Z}_p$ and returns $\mathcal{A}(param, g, g^u, g^v, g^{r_1}, g^{ur_2}, g^{vs})$.

We say that the *decisional linear (DLIN) assumption* holds for $\mathcal{G}_{DLIN}$ if for any probabilistic polynomial time algorithm $\mathcal{A}$, $\mathsf{Adv}^{dlin}_{\mathcal{A},G_{DLIN}}(\Lambda) \overset{def}{=} \big|\Pr[\mathbf{Exp}^{\mathsf{lin-1}}_{G_{DLIN},\mathcal{A}}(\Lambda) = 1] - \Pr[\mathbf{Exp}^{\mathsf{lin-2}}_{G_{DLIN},\mathcal{A}}(\Lambda) = 1]\big|$ is a negligible function of Λ.

3.2 A Construction TTBE1 of Threshold Tag-Based Encryption Scheme Based on the DBDH Assumption

Our first construction TTBE1 of threshold tag-based encryption scheme is obtained through a simplification and "thresholding" of the identity-based encryption scheme of Boneh and Boyen [1].

As easily seen, the identity-based encryption scheme of [1] can be simplified into a following tag-based encryption scheme. A public-key is randomly selected elements $g_1(= g^x), g_2, h_1$ on a bilinear group $\mathbb{G}$ (with generator g). The corresponding secret key is x. A message M is encrypted with respect to tag t as $(C, D, E) = (g^r, (g_1^t h_1)^r, M \cdot e(g_1, g_2)^r)$. Ciphertext (C, D, E) is decrypted with respect to tag t as $M = E/e(C, g_2)^x$ if it holds $e(C, g_1^t h_1) = e(D, g)$, otherwise $M = \perp$.

In the threshold identity-based encryption scheme of [2], which is also based on the identity-based scheme of [1], a decryption share is regarded as a ciphertext of private key share corresponding to decrypter's ID. On a while, in order to convert the above tag-based encryption scheme into a threshold version, thanks to the simple setting of tag-based encryption scheme, we can naturally distribute the secret key x into shares $\{f(i)\}_i$ using Shamir's secret sharing scheme [8] and make the i-th decryption share to be $(C^{f(i)}, E)$ as the usual threshold version of ElGamal encryption. More precisely TTBE1 is described in Figure 2.

```
Setup(n, k, Λ):
    $(p, g, \mathbb{G}, \mathbb{G}_1, e) \leftarrow G_{DBDH}(\Lambda)$;
    $x \xleftarrow{\$} \mathbb{Z}_p,\ f \xleftarrow{\$} \mathbb{Z}_p[X]$ satisfying $\deg(f) = k - 1$ and $f(0) = x$;
    $y, z \xleftarrow{\$} \mathbb{Z}_p,\ g_1 \leftarrow g^x,\ g_2 \leftarrow g^y,\ h_1 \leftarrow g^z$;
    $PK = (p, \mathbb{G}, \mathbb{G}_1, e, g, g_1, g_2, h_1),\ SK = (f(1), \cdots, f(n)),\ VK = (g^{f(1)}, \cdots, g^{f(n)})$;
    return $(PK, VK, SK)$.
Encrypt(PK, t, M):
    $r \xleftarrow{\$} \mathbb{Z}_p,\ C \leftarrow g^r,\ D \leftarrow ({g_1}^t h_1)^r,\ E \leftarrow M \cdot e(g_1, g_2)^r$, return $C_{tbe} = (C, D, E)$.
ShareDec(PK, i, SK_i = f(i), C_tbe = (C, D, ·), t):
    If $e(C, {g_1}^t h_1) \neq e(D, g)$ then return $\mu_i = (i, \perp)$ else return $\mu_i = (i, C^{f(i)})$.
ShareVf(PK, VK = (g^{f(i)}), C_tbe = (C, ·, ·), t, μ_i = (i, C_i)):
    If $e(C_i, g) \neq e(C, g^{f(i)})$ then return invalid else return valid.
Combine(PK, VK, C_tbe = (·, ·, E), t, {μ_1 = (1, C_1), ··· , μ_k = (k, C_k)}):
    If $\exists i$, ShareVf$(PK, VK_i, C_{tbe}, t, \mu_i)$ = invalid then return $\perp$,
     else return $E/e(\prod_{i=1}^{k} C_i^{\lambda_i}, g_2)$ using Lagrange coefficients $\lambda_1, \cdots, \lambda_k$
     satisfying $f(0) = \sum_{i=1}^{k} \lambda_i f(i)$.
```

Fig. 2. Threshold Tag-Based Encryption Scheme TTBE1

Theorem 2. *Under the DBDH assumption for G_{DBDH}, the threshold tag-based encryption scheme* $\mathsf{TTBE1}$ *is stag-CCA-secure.*

More precisely, for an arbitrary adversary $\mathcal{A}$ against stag-chosen-ciphertext security of $\mathsf{TTBE1}$ *that runs in time at most τ and makes at most Q decryption queries, there exists an algorithm $\mathcal{B}$ for the DBDH problem on G_{DBDH} that runs in time at most τ plus the time to perform $O(Q+n)$ exponentiations and $O(Q)$ pairing computations, and satisfies*

$$\mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE1},n,k}(\Lambda) = \mathsf{Adv}^{dbdh}_{\mathcal{B},G_{DBDH}}(\Lambda).$$

For an arbitrary adversary $\mathcal{A}'$ against stag decryption consistency of $\mathsf{TTBE1}$*, it holds that*

$$\mathsf{Adv}^{stag-dc}_{\mathcal{A}',\mathsf{TTBE1},n,k}(\Lambda) = 0.$$

Proof. First, we consider stag-chosen-ciphertext security of $\mathsf{TTBE1}$. Let $\mathcal{A}$ be an arbitrary adversary that runs in time at most τ, makes at most Q decryption queries, and has advantage $\mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE1},n,k}(\Lambda)$ in attacking $\mathsf{TTBE1}$ in the game of stag-chosen-ciphertext security. Using the adversary $\mathcal{A}$, we build an algorithm $\mathcal{B}$ that solves the DBDH problem on $G_{DBDH}(\Lambda)$.

Given $(\Lambda, p, \mathbb{G}, \mathbb{G}_1, e, g, g^a, g^b, g^c, W)$ as input, algorithm $\mathcal{B}$ proceeds as follows. (The aim of $\mathcal{B}$ is to distinguish two cases between $W = e(g,g)^{abc}$ or random.)

1. Initialization. Algorithm $\mathcal{B}$ invokes adversary $\mathcal{A}$ on input (n, k, Λ). Adversary $\mathcal{A}$ outputs a target tag t^* and a list $S = \{s_1, \cdots, s_{k-1}\} (\subset \{1, \cdots, n\})$ of the $k-1$ servers that it wishes to corrupt.
2. Setup. Then, $\mathcal{B}$ does the following:
   (a) $\mathcal{B}$ sets $g_1 = g^a, g_2 = g^b$ and computes $h_1 = g_1^{-t^*} g^{\gamma}$ with a random γ ($\xleftarrow{\$} \mathbb{Z}_p$). (This defines implicitly as $x = a, y = b, z = -t^* x + \gamma$.) $\mathcal{B}$ sets $PK = (p, \mathbb{G}, \mathbb{G}_1, e, g, g_1, g_2, h_1)$.
   (b) Next, $\mathcal{B}$ picks $k-1$ random integers $\alpha_1, \cdots, \alpha_{k-1} \xleftarrow{\$} \mathbb{Z}_p$. (We let $f \in \mathbb{Z}_p[X]$ be a polynomial of degree $k-1$ defined by $f(0) = x$ and $f(s_i) = \alpha_i$ for $i = 1, \cdots, k-1$. $\mathcal{B}$ does not know f.) $\mathcal{B}$ sets $SK|_S = (\alpha_1, \cdots, \alpha_{k-1})$.
   (c) For $i \in S$, $\mathcal{B}$ lets $u_i = g^{\alpha_i}$. For $i \notin S$, it computes $u_i = g_1^{\lambda_0} (g^{\alpha_1})^{\lambda_1} \cdots (g^{\alpha_{k-1}})^{\lambda_{k-1}}$, where $\lambda_0, \cdots, \lambda_{k-1} (\in \mathbb{Z}_p)$ are the Lagrange coefficients satisfying $f(i) = \lambda_0 f(0) + \sum_{j=1}^{k-1} \lambda_j f(s_j)$. (Note u_i satisfies $u_i = g^{f(i)}$.) $\mathcal{B}$ sets $VK = (u_1, \cdots, u_n)$.
   (d) $\mathcal{B}$ gives PK, VK and $SK|_S$ to $\mathcal{A}$.
3. Phase 1. $\mathcal{A}$ issues decryption share queries $((C_{tbe}, t), i)$ under the constraint that $t \neq t^*$ and $i \notin S$. First, $\mathcal{B}$ validates $e(C, g_1^t h_1) \overset{?}{=} e(D, g)$ to clarify the validity of ciphertext $C_{tbe} = (C, D, E)$. If validity test fails, $\mathcal{B}$ gives to $\mathcal{A}$ $(i, \perp)$. Otherwise, $\mathcal{B}$ computes the Lagrange coefficients $\lambda_1, \cdots, \lambda_{k-1}, \lambda_i \in \mathbb{Z}_p$ satisfying $f(0) = \lambda_i f(i) + \sum_{j=1}^{k-1} \lambda_j f(s_j)$ and sets

$$C_i = \left\{ \frac{\left(\frac{D}{C^{\gamma}}\right)^{\frac{1}{t-t^*}}}{C^{\sum_{j=1}^{k-1} \lambda_j \alpha_j}} \right\}^{\frac{1}{\lambda_i}}$$

and then gives to $\mathcal{A}$ (i, C_i) as decryption share. (If the ciphertext is valid, it must be the case that $C^{tx+z} = D$. Then, we have $D = C^{tx+z} = C^{tx+(-t^*x+\gamma)} = (C^x)^{t-t^*} C^\gamma$. Since $t \neq t^*$, we get $C^x = (D/C^\gamma)^{\frac{1}{t-t^*}}$. Then, substituting $f(0) = \sum_{j=1}^{k-1} \lambda_j f(s_j) + \lambda_i f(i)$ for x, and noting $C_i = C^{f(i)}$ we obtain the above expression of C_i.)

4. Challenge. $\mathcal{A}$ outputs two same-length messages M_0 and M_1. $\mathcal{B}$ flips a fair coin $b \in \{0, 1\}$, and responds with the challenge ciphertext $C^*_{tbe} = (g^c, (g^c)^\gamma, M_b W)$. (As $\mathcal{B}$ sets $h_1 = g_1^{-t^*} g^\gamma$, it holds that $(g^c)^\gamma = (h_1 g_1^{t^*})^c$. Moreover, if $W = e(g, g)^{abc}$, then we have $M_b \cdot W = M_b \cdot e(g_1, g_2)^c$ and C^*_{tbe} is a valid ciphertext of M_b under PK with tag t^*.)
5. Phase 2. $\mathcal{A}$ issues additional queries as in Phase 1, to which $\mathcal{B}$ responds as before.
6. Guess. Eventually, $\mathcal{A}$ outputs a guess b'. $\mathcal{B}$ outputs 1 if $b = b'$, or outputs 0 otherwise.

This completes the description of algorithm $\mathcal{B}$, that runs in time at most τ plus the time to perform $O(Q + n)$ exponentiations and $O(Q)$ pairing computations. By the comments in the description, it is immediate that $\mathcal{B}$ perfectly simulates a stag-CCA game for $\mathcal{A}$ if $W = e(g, g)^{abc}$. When W is a random element, the view of $\mathcal{B}$ is independent of the choice of b. So, $\mathsf{Adv}^{dbdh}_{\mathcal{B},G_{DBDH}}(\Lambda) = |\Pr[b = b'] - 1/2| = |(1/2 + \mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE1},n,k}(\Lambda)) - 1/2| = \mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE1},n,k}(\Lambda)$.

Second, we consider stag decryption consistency of $\mathsf{TTBE1}$. Let $\mathcal{A}'$ be an arbitrary adversary with advantage $\mathsf{Adv}^{stag-dc}_{\mathcal{A}',\mathsf{TTBE1},n,k}(\Lambda)$ in attacking $\mathsf{TTBE1}$ in the game of stag decryption consistency. Suppose adversary $\mathcal{A}'$ outputs $t, C_{tbe}, S = (\mu_1 = (1, C_1), \cdots, \mu_k = (1, C_k)), S' = (\mu'_1 = (1, C'_1), \cdots, \mu'_k = (k, C'_k))$. If those shares μ_i in S are valid, they must satisfy $e(C_i, g) = e(C, g^{f(i)})$, so $C_i = C^{f(i)}$. Then, it holds that $\prod_{i=1}^k C_i^{\lambda_i} = C^{\sum_{i=1}^k \lambda_i f(i)} = C^x$. Similarly, if the shares μ'_i in S' are valid, we have $\prod_{i=1}^k {C'}_i^{\lambda_i} = C^x$. This means $\mathsf{Combine}(PK, VK, C_{tbe}, t, S) = \mathsf{Combine}(PK, VK, C_{tbe}, t, S')$. Thus, $\mathsf{Adv}^{stag-dc}_{\mathcal{A}',\mathsf{TTBE1},n,k}(\Lambda) = 0$. □

3.3 A Construction TTBE2 of Threshold Tag-Based Encryption Scheme Based on the DLIN Assumption

Our second construction $\mathsf{TTBE2}$ of threshold tag-based encryption scheme naturally expands the Kiltz's tag-based encryption scheme [6] to the threshold setting. A secret key x_1, x_2 of the Kiltz's scheme is distributed among n secret key shares $(f_1(1), f_2(1)), \cdots, (f_1(n), f_2(n))$ with polynomials f_1, f_2 satisfying $x_1 = f_1(0), x_2 = f_2(0)$. Decryption shares are of the form $(C_1^{f_1(i)}, C_2^{f_2(i)})$.

More precisely $\mathsf{TTBE2}$ is described in Figure 3.

Theorem 3. *Under the DLIN assumption for G_{DLIN}, the threshold tag-based encryption scheme* $\mathsf{TTBE2}$ *is stag-CCA-secure.*

More precisely, for an arbitrary adversary $\mathcal{A}$ against stag-chosen-ciphertext security of $\mathsf{TTBE2}$ *that runs in time at most τ and makes at most Q decryption*

```
$\mathsf{Setup}(n, k, \Lambda)$:
  $(p, g, \mathbb{G}, \mathbb{G}_1, e) \leftarrow G_{DLIN}(\Lambda); \quad x_1, x_2 \xleftarrow{\$} \mathbb{Z}_p;$
  $f_1, f_2 \xleftarrow{\$} \mathbb{Z}_p[X]$ satisfying $\deg(f_1) = \deg(f_2) = k - 1$ and $f_1(0) = x_1, f_2(0) = x_2;$
  $z \leftarrow g_1^{x_1},\ g_2 \leftarrow z^{\frac{1}{x_2}},\ y_1, y_2 \xleftarrow{\$} \mathbb{Z}_p,\ u_1 \leftarrow g_1^{y_1}, u_2 \leftarrow g_2^{y_2};$
  $PK = (p, \mathbb{G}, \mathbb{G}_1, e, g_1, g_2, z, u_1, u_2),\ SK = ((f_1(1), f_2(1)), \cdots, (f_1(n), f_2(n)));$
  $VK = ((v_{11} = g_1^{f_1(1)}, v_{12} = g_2^{f_2(1)}), \cdots, (v_{n1} = g_1^{f_1(n)}, v_{n2} = g_2^{f_2(n)}));$
  return $(PK, VK, SK)$.
$\mathsf{Encrypt}(PK, t, M)$:
  $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p,\ C_1 \leftarrow g_1^{r_1},\ C_2 \leftarrow g_2^{r_2},\ D_1 \leftarrow (z^t u_1)^{r_1},\ D_2 \leftarrow (z^t u_2)^{r_2},\ E \leftarrow M z^{r_1 + r_2};$
  return $C_{tbe} = (C_1, C_2, D_1, D_2, E)$.
$\mathsf{ShareDec}(PK, i, SK_i = (f_1(i), f_2(i)), C_{tbe} = (C_1, C_2, D_1, D_2, \cdot), t)$:
  If $e(C_1, z^t u_1) \neq e(D_1, g_1)$ or $e(C_2, z^t u_2) \neq e(D_2, g_2)$ then return $\mu_i = (i, \perp)$,
   else return $\mu_i = (i, (C_1^{f_1(i)}, C_2^{f_2(i)}))$.
$\mathsf{ShareVf}(PK, VK = ((v_{i1}, v_{i2})), C_{tbe} = (C_1, C_2, \cdot, \cdot, \cdot), t, \mu_i = (i, (C_{i1}, C_{i2})))$ :
  If $e(C_{i1}, g_1) \neq e(C_1, v_{i1})$ or $e(C_{i2}, g_2) \neq e(C_2, v_{i2})$ then return invalid, else return valid.
$\mathsf{Combine}(PK, VK, C_{tbe} = (\cdot, \cdot, \cdot, \cdot, E), t, \{\mu_1 = (1, (C_{11}, C_{12})), \cdots, \mu_k = (k, (C_{k1}, C_{k2}))\})$:
  If $\exists i,\ \mathsf{ShareVf}(PK, VK, C_{tbe}, t, \mu_i) = \mathsf{invalid}$ then return $\perp$,
   else return $E / \prod_{i=1}^{k} (C_{i1} C_{i2})^{\lambda_i}$ using Lagrange coefficients $\lambda_1, \cdots, \lambda_k$
   satisfying $f_1(0) = \sum_{i=1}^{k} \lambda_i f_1(i)$.
```

Fig. 3. Threshold Tag-Based Encryption Scheme TTBE2

queries, there exists an algorithm $\mathcal{B}$ for the DLIN problem on G_{DLIN} that runs in time at most τ plus the time to perform $O(Q + n)$ exponentiations and $O(Q)$ pairing computations and satisfies

$$\mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE2},n,k}(\Lambda) = \mathsf{Adv}^{dbdh}_{\mathcal{B},G_{DLIN}}(\Lambda).$$

For an arbitrary adversary $\mathcal{A}'$ against stag decryption consistency of TTBE2*, it holds that*

$$\mathsf{Adv}^{stag-dc}_{\mathcal{A}',\mathsf{TTBE2},n,k}(\Lambda) = 0.$$

Proof. First, we consider stag chosen ciphertext security of TTBE2. Let $\mathcal{A}$ be an arbitrary adversary that runs in time at most τ, makes at most Q decryption queries, and has advantage $\mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE2},n,k}(\Lambda)$ in attacking TTBE2 in the game of stag-chosen-ciphertext security. Using the adversary $\mathcal{A}$, we build an algorithm $\mathcal{B}$ that solves the DLIN problem of $G_{DLIN}(\Lambda)$.

Given $(\Lambda, p, \mathbb{G}, \mathbb{G}_1, e, g_1, g_2, z, g_1^{r_1}, g_2^{r_2}, W)$ as input, algorithm $\mathcal{B}$ proceeds as follows. (The aim of $\mathcal{B}$ is to distinguish two cases between $W = z^{r_1 + r_2}$ or random.)

1. Initialization. Algorithm $\mathcal{B}$ invokes adversary $\mathcal{A}$ on input (n, k, Λ). Adversary $\mathcal{A}$ outputs a target tag t^* and a list $S = \{s_1, \cdots, s_{k-1}\}(\subset \{1, \cdots, n\})$ of the $k - 1$ servers that it wishes to corrupt.
2. Setup. Then, $\mathcal{B}$ does the following:
   (a) $\mathcal{B}$ picks random integers $c_1, c_2 \xleftarrow{\$} \mathbb{Z}_p$ and computes $u_1 = z^{-t^*} g_1^{c_1},\ u_2 = z^{-t^*} g_2^{c_2}$. (This defines implicitly as $y_1 = -t^* x_1 + c_1,\ y_2 = -t^* x_2 + c_2$.) $\mathcal{B}$ sets $PK = (p, \mathbb{G}, \mathbb{G}_1, e, g_1, g_2, z, u_1, u_2)$.

   (b) Next, $\mathcal{B}$ picks $2k-2$ random integers $\alpha_1, \cdots, \alpha_{k-1}, \beta_1, \cdots, \beta_{k-1} \xleftarrow{\$} \mathbb{Z}_p$. (We let $f_1, f_2 \in \mathbb{Z}_p[X]$ be two polynomials of degree $k-1$ defined by $f_1(0) = x_1, f_1(s_i) = \alpha_i$ $(i = 1, \cdots, k-1)$ and $f_2(0) = x_2, f_2(s_i) = \beta_i$ $(i = 1, \cdots, k-1)$. $\mathcal{B}$ does not know f_1, f_2.) $\mathcal{B}$ sets $SK|_S = (SK_{s_1} = (\alpha_1, \beta_1), \cdots, SK_{s_{k-1}} = (\alpha_{k-1}, \beta_{k-1}))$.

   (c) For $i \in S$, $\mathcal{B}$ lets $VK_i = (v_{i1}, v_{i2}) = (g_1^{\alpha_i}, g_2^{\beta_i})$. For $i \notin S$, it computes $v_{i1} = z^{\lambda_0}(g_1^{\alpha_1})^{\lambda_1} \cdots (g_1^{\alpha_{k-1}})^{\lambda_{k-1}}$ and $v_{i2} = z^{\lambda_0}(g_2^{\beta_1})^{\lambda_1} \cdots (g_2^{\beta_{k-1}})^{\lambda_{k-1}}$, where $\lambda_0, \cdots, \lambda_{k-1} (\in \mathbb{Z}_p)$ are the Lagrange coefficients satisfying $f(i) = \lambda_0 f(0) + \sum_{j=1}^{k-1} \lambda_j f(s_j)$ for degree k polynomials f. (As easily seen, (v_{i1}, v_{i2}) satisfies $v_{i1} = g_1^{f_1(i)}, v_{i2} = g_2^{f_2(i)}$.) $\mathcal{B}$ sets $VK = (VK_1 = (v_{11}, v_{12}), \cdots, VK_n = (v_{n1}, v_{n2}))$.

   (d) $\mathcal{B}$ gives PK, VK and $SK|_S$ to $\mathcal{A}$.

3. Phase 1. $\mathcal{A}$ issues decryption share queries $((C_{tbe}, t), i)$ under the constraint that $t \neq t^*$ and $i \notin S$. First, $\mathcal{B}$ validates $e(C_1, z^t u_1) \overset{?}{=} e(D_1, g_1)$ and $e(C_2, z^t u_2) \overset{?}{=} e(D_2, g_2)$ to clarify validity of the ciphertext $C_{tbe} = (C_1, C_2, D_1, D_2, E)$. If validity test fails, $\mathcal{B}$ gives to $\mathcal{A}$ $(i, \perp)$. Otherwise, $\mathcal{B}$ computes the Lagrange coefficients $\lambda_1, \cdots, \lambda_{k-1}, \lambda_i \in \mathbb{Z}_p$ satisfying $f(0) = \lambda_i f(i) + \sum_{j=1}^{k-1} \lambda_j f(s_j)$ for degree k polynomials f and sets

$$C_{i1} = \left\{ \frac{\left(\frac{D_1}{C_1^{c_1}}\right)^{\frac{1}{t-t^*}}}{C_1^{\sum_{j=1}^{k-1} \lambda_j \alpha_j}} \right\}^{\frac{1}{\lambda_i}}, \quad C_{i2} = \left\{ \frac{\left(\frac{D_2}{C_2^{c_2}}\right)^{\frac{1}{t-t^*}}}{C_2^{\sum_{j=1}^{k-1} \lambda_j \beta_j}} \right\}^{\frac{1}{\lambda_i}}$$

   and then gives to $\mathcal{A}$ $(i, (C_{i1}, C_{i2}))$ as decryption share. (If the ciphertext is valid, it must be the case that $D_1 = C_1^{(t-t^*)x_1 + c_1} = (C_1^{x_1})^{t-t^*} C_1^{c_1}$. Then, since $t \neq t^*$, we get $C_1^{x_1} = (D_1/C_1^{c_1})^{\frac{1}{t-t^*}}$. Substituting $f_1(0) = \sum_{j=1}^{k-1} \lambda_j f_1(s_j) + \lambda_i f_1(i)$ for x_1, and noting $C_{1i} = C_1^{f_1(i)}$, we obtain the above expression of C_{i1}. Similar for C_{i2}.)

4. Challenge. $\mathcal{A}$ outputs two same-length messages M_0 and M_1. $\mathcal{B}$ flips a fair coin $b \in \{0, 1\}$, and responds with the challenge ciphertext $C_{tbe}^* = (g_1^{r_1}, g_2^{r_2}, (g_1^{r_1})^{c_1}, (g_2^{r_2})^{c_2}, M_b W)$. (As $\mathcal{B}$ sets $u_1 = z^{-t^*} g_1^{c_1}$ and $u_2 = z^{-t^*} g_2^{c_2}$, it holds that $(g_1^{r_1})^{c_1} = (u_1 z^{t^*})^{r_1}, (g_2^{r_2})^{c_2} = (u_2 z^{t^*})^{r_2}$. Moreover, if $W = z^{r_1 + r_2}$, then we have $M_b W = M_b z^{r_1 + r_2}$ and C_{tbe}^* is a valid ciphertext of M_b under PK with tag t^*.)
5. Phase 2. $\mathcal{A}$ issues additional queries as in Phase 1, to which $\mathcal{B}$ responds as before.
6. Guess. Eventually, $\mathcal{A}$ outputs a guess b'. $\mathcal{B}$ outputs 1 if $b = b'$, or outputs 0 otherwise.

This completes the description of algorithm $\mathcal{B}$, that runs in time at most τ plus the time to perform $O(Q+n)$ exponentiations and $O(Q)$ pairing computations. By the comments in the description, it is immediate that $\mathcal{B}$ perfectly simulates a stag-CCA game for $\mathcal{A}$ if $W = z^{r_1 + r_2}$. When W is a random element, the view of $\mathcal{B}$ is independent of the choice b. So, $\mathsf{Adv}^{dlin}_{\mathcal{B},G_{DLIN}}(\varLambda) = |\Pr[b = b'] - 1/2| = |(1/2 + \mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE2},n,k}(\varLambda)) - 1/2| = \mathsf{Adv}^{stag-cca}_{\mathcal{A},\mathsf{TTBE2},n,k}(\varLambda)$.

Second, we consider stag decryption consistency of $\mathsf{TTBE2}$. Let $\mathcal{A}'$ be an arbitrary adversary with advantage $\mathsf{Adv}^{stag-dc}_{\mathcal{A}',\mathsf{TTBE2},n,k}(\varLambda)$ in attacking $\mathsf{TTBE2}$ in the game of stag decryption consistency. Suppose adversary $\mathcal{A}'$ outputs t, C_{tbe}, $S = (\mu_1 = (1, (C_{11}, C_{12})), \cdots, \mu_k = (k, (C_{k1}, C_{k2})))$, $S' = (\mu'_1 = (1, (C'_{11}, C'_{12})), \cdots, \mu'_k = (k, (C'_{k1}, C'_{k2})))$. If those shares μ_i in S are valid, they must satisfy $e(C_{i1}, g_1) = e(C_1, g_1^{f_1(i)})$, $e(C_{i2}, g_2) = e(C_2, g_2^{f_2(i)})$, so $C_{i1} = C_1^{f_1(i)}$, $C_{i2} = C_2^{f_2(i)}$. Then, it holds that $\prod_{i=1}^{k} C_{i1}^{\lambda_i} = C_1^{\sum_{i=1}^{k} \lambda_i f_1(i)} = C_1^{x_1}$ and $\prod_{i=1}^{k} C_{i2}^{\lambda_i} = C_2^{\sum_{i=1}^{k} \lambda_i f_2(i)} = C_2^{x_2}$. Similarly, if the shares μ'_i in S' are valid, we have $\prod_{i=1}^{k} {C'}_{i1}^{\lambda_i} = C_1^{x_1}$ and $\prod_{i=1}^{k} {C'}_{i2}^{\lambda_i} = C_2^{x_2}$. This means $\mathsf{Combine}(PK, VK, C_{tbe}, t, S) = \mathsf{Combine}$ (PK, VK, C_{tbe}, t, S'). Thus, $\mathsf{Adv}^{stag-dc}_{\mathcal{A}',\mathsf{TTBE2},n,k}(\varLambda) = 0$. □

4 Construction of Threshold Public Key Encryption Schemes

By applying the conversion TT2TP in Section 2 to the two threshold tag-based encryption schemes $\mathsf{TTBE1}, \mathsf{TTBE2}$ in Section 3, we obtain two threshold public key encryption schemes $\mathsf{TPKE1}, \mathsf{TPKE2}$ that are both CCA-secure by Theorem 1, Theorem 2 and Theorem 3:

Theorem 4. *Let* S *be a strong one-time signature.*

- *Under the DBDH assumption,* $\mathsf{TPKE1} = \mathit{TT2TP}(\mathsf{TTBE1}, \mathsf{S})$ *is a CCA-secure threshold public key encryption scheme.*
- *Under the DLIN assumption,* $\mathsf{TPKE2} = \mathit{TT2TP}(\mathsf{TTBE2}, \mathsf{S})$ *is a CCA-secure threshold public key encryption scheme.*

Our threshold public key encryption schemes TPKE1 and TPKE2 are more simple than the construction BBH given by Boneh, Boyen and Halevi [2]. It is because our constructions are based on the tag-based schemes which is more simple than the ID-scheme used by BBH.

Instantly, Table 1 shows a comparison of efficiency among TPKE1, TPKE2 and BBH. The table shows the number of pairings, multi-exponentiations and regular-exponentiations required in operations of those schemes. (The entries of

Table 1. Efficiency Comparison among TPKE1, TPKE2 and BBH

Scheme	Assumption	Encrypt	ShareDec	ShareVf	Combine	reduction
		♯pairings + [♯multi−exp., ♯reg−exp.]				
BBH	DBDH	$1 + [1, 3]$	$2 + [1, 2]$	$2 + [0, 1]$	$2 + [2, 0]$	tight
TPKE1	DBDH	$1 + [1, 3]$	$2 + [0, 2]$	$2 + [0, 0]$	$1 + [1, 0]$	tight
TPKE2	DLIN	$0 + [2, 3]$	$4 + [0, 3]$	$4 + [0, 0]$	$0 + [1, 0]$	tight

Combine do not include costs for ShareVf.) As shown, TPKE1 is more efficient than BBH. TPKE1 requires less number of exponentiation both in ShareDec, ShareVf and Combine than BBH. On the other hand, TPKE2 has an advantage that it is most efficient both in Encrypt and Combine because it requires no computation of bilinear map in those operations.

5 Conclusion

In this paper, we proposed a notion of threshold tag-based encryption schemes and showed a conversion from any stag-CCA-secure threshold tag-based encryption schemes to CCA-secure threshold public-key encryption schemes. We gave two constructions of threshold tag-based encryption schemes and obtained two constructions of threshold public-key encryption schemes, using that conversion. Those schemes are non-interactive, robust, proved secure without random oracle model and are more efficient than the construction by Boneh, Boyen and Halevi [2].

References

1. 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)
2. Boneh, D., Boyen, X., Halevi, S.: Chosen ciphertext secure public key threshold encryption without random oracles. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 226–243. Springer, Heidelberg (2006)
3. 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)
4. Canetti, R., Goldwasser, S.: An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 90–106. Springer, Heidelberg (1999)
5. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004)
6. Kiltz, E.: Chosen-ciphertext security from tag-based encryption. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 581–600. Springer, Heidelberg (2006)
7. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairings. In: Proceedings of the Symposium on Cryptography and Information Security, SCIS 2000, Japan (2000)
8. Shamir, A.: How to share a secret. Communications of the ACM, 612–613 (1979)
9. Shoup, V., Gennaro, R.: Securing threshold cryptosystems against chosen ciphertext attack. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 1–16. Springer, Heidelberg (1998)

Malyzer: Defeating Anti-detection for Application-Level Malware Analysis

Lei Liu and Songqing Chen

Department of Computer Science
George Mason University
{lliu3,sqchen}@cs.gmu.edu

Abstract. Malware analysis is critical for malware detection and prevention. To defeat malware analysis and detection, today malware commonly adopts various sophisticated anti-detection techniques, such as performing debugger, emulator, and virtual machine fingerprinting, and camouflaging its traffic as normal legitimate traffic. These mechanisms produce more and more stealthy malware that greatly challenges existing malware analysis schemes.

In this work, targeting application level stealthy malware, we propose Malyzer, *the key of which is to defeat malware anti-detection mechanisms at startup and runtime so that malware behavior during execution can be accurately captured and distinguished.* For analysis, Malyzer always starts a copy, referred to as a shadow process, of any suspicious process on the same host by defeating all startup anti-detection mechanisms employed in the process. To defeat internal runtime anti-detection attempts, Malyzer further makes this shadow process mutually invisible to the original suspicious process. To defeat external anti-detection attempts, Malyzer makes as if the shadow process runs on a different machine to the outside. Since ultimately malware will conduct local information harvesting or dispersion, Malyzer constantly monitors the shadow process's behavior and adopts a hybrid scheme for its behavior analysis. In our experiments, Malyzer can accurately detect all malware samples that employ various anti-detection techniques.

1 Introduction

Internet malware poses an immense threat to computer system security. Fundamentally, malware aims to collect local sensitive information, such as bank account information, password, and CD keys, or leverage infected hosts for various attacks, such as spam relay, DDoS, IP laundering (acting as stepping stones), and phishing. These malicious actions are commonly referred to as information harvesting and information dispersion [15], respectively.

A number of schemes have been proposed and used for malware detection and analysis. Among them, the signature-based approach has been employed for many years and is the most prevalent scheme in practice. In general, signature-based schemes [8,19,20,28] generate content-based signatures that can uniquely identify the malware. Signature-based schemes are efficient and effective in detecting and containing known malware, but they are inherently ineffective against

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 201–218, 2009.

previously unknown or polymorphic/metamorphic malware [23]. Encryption [10] and obfuscation [25] are also commonly used to make the signature-based schemes incompetent.

Research has been conducted on behavior analysis that complements the content-based signature approach. Behavior analysis aims to identify abnormal process behavior. In general, the process behavior under supervision is compared with a pre-defined safe model. Any behavior deviating from the predefined model would lead to an alarm, which is in line with the anomaly-based approach. For behavior analysis, various approaches have been studied, such as using dynamic taint processing [11,12,24,29,31], checking auto-start extensibility points in registry [30], and searching for various hooks [9,27].

However, due to underlying economic motivations, various anti-detection mechanisms have been constantly and continuously developed and quickly adopted by malware developers to make more and more stealthy malware. The efforts are from two perspectives. One is to use protective camouflaging to conceal its existence. For example, modern bots try to blend their traffic with normal user traffic [16,21]. Some malware [1] may run as browser helper objects (BHO). Advanced malware [2] is found to perform dynamic code replacement. That is, a benign user application process is started, and malware code is then written to its memory sections and executed in the context of this process. When misbehavior is identified, it will be traced down to a "legitimate" application process. This approach becomes more and more common since malware can easily go through firewalls via such an approach.

Besides camouflaging at runtime, malware developers today also commonly adopt proactive approaches to evade detection at startup. For example, as malware analysis and detection may use various debugger, emulator, or run suspicious code samples in a virtual machine environment, most of today's malware performs virtual machine, emulator, and debugger detection, which we refer to as running environment tests, before the logic of malware gets executed [3].[1] These techniques make systems like Panorama [31] (that relies on an emulator for malware analysis) and SpyProxy [22] (that executes the Web content in a virtual machine) to stop functioning. Widely adopting these anti-detection techniques in malware, malware developers successfully enforce malware analysis and detection to be conducted in a real executing environment, in which various runtime camouflaging as aforementioned can effectively protect malware.

Targeting application-level stealthy malware, in this paper, we propose Malyzer, an execution-based approach for malware analysis. *The key idea of Malyzer is to unveil malware camouflaging at startup and runtime so that malware behavior can be accurately captured and distinguished.* For this purpose, Malyzer constantly monitors processes' startup procedures. If a process is suspicious (i.e., analysis object), Malyzer thus can start a copy of this process, which is referred to as a shadow process, on the same host no matter what anti-detection techniques have been employed at startup. Malyzer makes the shadow process inaccessible

[1] Such tests could be combined into a packer that can perform multi-layer packing for anti-reverse-engineering [17] against static malware code analysis.

to other processes or users in order to eliminate noisy activities that do not belong to the shadow process. Without interferences caused by other processes or users on the same host, the behavior of the shadow process could only be autonomous by its inner logic or caused by some remote control.

Because the shadow process runs in the same environment as the original suspicious process, the environment test including virtual machine detection , should that be performed, will always pass. To defeat internal anti-detection tests, Malyzer further makes this shadow process mutually invisible to the original process. Malyzer also controls direct user accesses to this shadow process in order to minimize legitimate user activities that malware could leverage for camouflaging. To defeat external anti-detection attempts, Malyzer makes the shadow process running as if it is running on a different machine to the outsider. Since ultimately malware will conduct local information collection or dispersion, Malyzer constantly monitors the shadow process's disk, network, and memory accesses and uses a hybrid scheme combing both anomaly-based and signature-based approaches for process behavior analysis. We have implemented a prototype system of Malyzer. Our evaluation results show that it can accurately capture the behavior of all the malware samples that use various anti-detection techniques in our experiments.

The rest of the paper is organized as follows. We present Malyzer design in section 2 and prototype implementation in section 3. Malyzer is evaluated in section 4. We further discuss some optimizations and limitations of Malyzer in section 5 and make concluding remarks in section 6.

2 Malyzer Design

Figure 1 shows the system architecture of Malyzer design. Malyzer consists of three components: `Startup Tracker`, `Shadow Process Manager`, and `Shadow Process Monitor`.

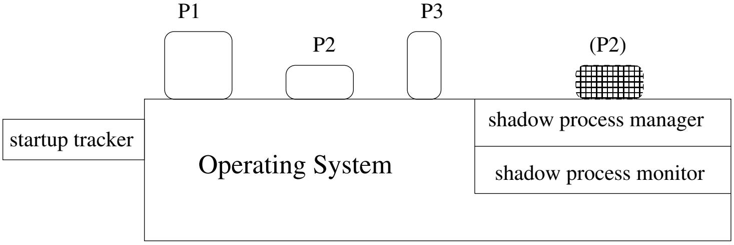


Fig. 1. System Architecture. Suppose the process P2 is suspicious. Through `Startup Tracker`, information regarding how P2 is started is fetched as well as a memory image of P2 (if necessary). No matter how P2 is started, a shadow process of P2 could be started on the same host by `Shadow Process Manager`, which defeats all kinds of anti-detections. Then `Shadow Process Monitor` monitors the disk/memory/network accesses of the shadow process for malware behavior analysis.

2.1 Startup Tracker

To start a shadow process of the given suspicious process, Malyzer must obtain the runnable executable of the malware so that a shadow process can be started on demand. With sophisticated anti-detection mechanisms, such as dynamic code replacement, however, obtaining the executable of a given process is sometimes non-trivial. For example, directly dumping the memory image of the given process often does not work.

In general, today malware runs in three possible forms and there are three types of relationships between a malware process and the corresponding executable.

- Malware runs directly from the executable (possibly with multi-layer packing) on the disk. This is the most common and trivial approach. The malware process starts when the system starts (usually it is achieved by modifying registry entries). Given a process, it is easy to locate its executable on the disk by querying the module name, through which the full path of the executable can be obtained. Many traditional virus and worms use this approach.
- Malware runs as a DLL in the context of a benign application process. In this approach, the malware is encapsulated into a DLL, and then installed statically to a benign application or dynamically to a running process. For example, when a user browses a particular Web site, the user is fooled to install a helper object to the Web browser. `WebBuying` [1] uses this approach.
- Malware runs through dynamic code replacement. This is an advanced and prevailing approach. The common thread of this approach is as follows:
  1. The malware first starts a benign process in a suspended mode. The system API `createProcess`[2] is normally used to invoke the benign process.
  2. The malware then allocates memory in the domain of the suspended process and injects its own executable. This is often conducted through `writeProcessMemory`.
  3. The malware then sets entry point and resumes execution of suspended process by using `resumeThread` or `createRemoteThread`. The logic of benign process is completely skipped.

  Figure 2 shows an example of dynamic code replacement of graybird, which is one of the most prolific piece of Windows malware [2]. With wide usage of this approach, querying the module name only returns the genuine executable of a benign process on the disk, if an identified suspicious process is given. This not only allows the malware to go through firewalls which is a crucial design target of modern malware, but also misleads malware analysis and detection.

With these three approaches and the commonly adopted other anti-detection mechanisms, it is difficult 1) to locate the runnable malware executable; 2) even if a malware code sample is identified, it may not be start-able.

[2] A family of APIs can be used for process creation, such as `createProcessAsUser` and `createProcessWithLogonW`. We use `createProcess` to represent all of these. We take a similar approach for other APIs for brevity.

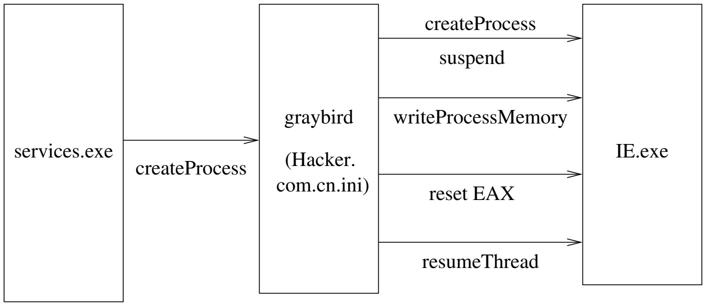


Fig. 2. Dynamic Code Replacement of `graybird`. `graybird` camouflages itself as Internet Explorer (IE). It starts an IE process, and then replaces the memory image of IE with graybird. After resetting EAX, it resumes the process to execute the real malcode of graybird, but appears to be an IE process in the system.

Dynamic code replacement/injection is not uncommon in normal programming practice. With the increasing use of encryption technologies and packers, dynamic code replacement has been used by a lot of benign applications for all kinds of purposes, such as code protection [14]. But we noticed that dynamic code replacement adopted by benign applications is usually confined within the application processes, and it's rare that dynamic code replacement requires to start another process in these benign applications.

Therefore, `Startup Tracker` is designed to differentiate these situations and to preserve critical startup information for `Shadow Process Manager`. In addition, `Startup Tracker` can also provide hints for malware detection. For example, a benign process would rarely start with a **inter-process** dynamic code replacement approach.

Running as a driver to the host operating system, `Startup Tracker` tracks the process creation procedure once the machine is started. In Windows systems, Windows does not provide a convenient mechanism to track process creation and termination. Malyzer thus keeps listening to all process creation notification messages. A notification message typically includes information such as a process ID and a parent process ID. In Unix-like systems, such information can be easily obtained from the process data structure.

To identify different process startup approaches, `Startup Tracker` also needs API level information to discover possible dynamic code replacement. `Startup Tracker` achieves this through identifying a unique API call sequence. That is, as aforementioned, for dynamic code replacement, the successive system API calls of `createProcess`, `writeProcessMemory`, and `ResumeThread` are inevitable. Thus, when `Startup Tracker` monitors the process startup procedure, if a sequence of the above system API calls with appropriate parameters is identified, it is highly likely to be dynamic code replacement, which is suspicious.

In addition, `Startup Tracker` also needs to prepare for starting a shadow process of any identified suspicious process as we discuss in the next section. Therefore, if dynamic code replacement is found, `Startup Tracker` also dumps the process initial memory image before the process is resumed as well as recording other information, such as section sizes and the entry point.

If a malware process runs directly from the executable on the disk or runs as a DLL of a benign application, `Startup Tracker` can provide the child-parent information regarding how a process is started. If the malware runs as a DLL to a benign application, we will discuss, in the next section, how to match up the DLL(s) in the shadow process at runtime.

2.2 Shadow Process Manager

The role of `Shadow Process Manager` is mainly to defeat runtime malware anti-detection mechanisms that come from the internal or external sources. Thus, `Shadow Process Manager` needs to make the shadow process to run as a normal process, which is mutually invisible to the original process to deal with internal anti-detection mechanisms, while it is accessible to the outside to deal with any external anti-detections.

2.2.1 Defeating Internal Anti-detections: The Shadow Process Is Mutually Invisible to the Original Process

With the help of `Startup Tracker` that has done sufficient preparation, `Shadow Process Manager` can start the shadow process with ease. However, `Shadow Process Manager` also needs to guarantee a normal status of the shadow process at runtime since various techniques may be used by a malware instance to perform anti-detections during its execution. For example, a simple but commonly used detection performed by a malware process is to frequently check if there are multiple instances of itself running on the same host. If there are, the process terminates. In our experiments, nearly all malware samples adopt some mechanisms to prevent multiple instances from running on the same host. To defeat these anti-detection mechanisms, `Shadow Process Manager` thus needs to use an uncommon approach to start the shadow process, and hook up some interceptors when the shadow process is started.

To detect multiple instances, usually a process could use the following mechanisms:

- **through shared memory section:** Statement like $\#pragma\ comment(linker, ``/section : SHARED, RWS'')$ can set up a shared section between different instances of the same process. A process can access the section directly and detect if it has been modified by other process instance. It is also possible to create a shared section with API like `NtCreateSection` or `CreateFileMapping` at run time. Different process instances can only access the shared section through a name or file handle. So we treat this case as a named object.
- **through process list checking:** The process can emulate the process list and check if there is any process with the same module name as itself. This approach is not used if the malware runs through DLL injection.

- **through named object:** Named objects provide a convenient approach for processes to share object handles. After a process has created a named event, mutex, semaphore, file-mapping, section or timer object, other processes can use the name to open the handle to the object. If a process tries to create an object using a name that is in use by another process, the function fails and `GetLastError` returns `ERROR_INVALID_HANDLE`. In general, mutex is the most commonly used named object.

Among these mechanisms, accessing shared memory section cannot be intercepted. If a memory section is shared, the section has the corresponding `SHARED` property. When the operating system (OS) loads this section, the OS checks whether the section exists. If it does, the OS only points the section to the existing one so that multiple process instances will access the same memory region.

One possible solution to deal with this in the shadow process (in order to prevent possible multiple copy detection) is to copy the portable executable (PE) to a file with a different name. The solution is simple and effective but it also has limitations. In our experiments, all malware samples query the module name, and proceed according to the query result. Although it is possible to add additional interceptors to change the module name, it introduces new complexity.

Thus, `Shadow Process Manager` takes another approach, similar to dynamic code replacement used by malware, in which the PE content is read, aligned, and copied to a suspended process. In this procedure, however, the copy does not honor the `SHARED` property of memory sections. It simply allocates new memory space for all sections and then copies everything. Thus, the shared section would present as a new section in the memory [18]. The suspended process, which we refer to as a shell, could be any process with respect to defeating the memory sharing approach.

Considering the possible usage of shared memory section, `Shadow Process Manager` thus always starts the shadow process using a shell. Since the original suspicious process is running in the system, its shell PE must exist. Malyzer can always start a shell from this PE with the right module name. Then

- If no dynamic code replacement is found by `Startup Tracker`, `Shadow Process Manager` will launch the shadow process based on the portable executable (PE) on the disk. That is, `Startup Process Manager` reads the PE, aligns, and writes to the memory.
- If dynamic code replacement is found, `Shadow Process Manager` will start the shadow process based on the dumped memory image by `Startup Tracker`. That is, `Shadow Process Manager` will create the shell, then copy the memory image into the process and resume the execution.
- In either of the above two situations, some malcode may be injected as a DLL into the process dynamically. `Shadow Process Manager` needs to guarantee that the shadow process load all DLLs as the original suspicious process. Therefore, after a shadow process is launched, Malyzer further compares the DLL list of the shadow process with that in the original suspicious process. If any DLL is missing in the shadow process, `Shadow Process Manager` will insert the corresponding DLL into the shadow process.

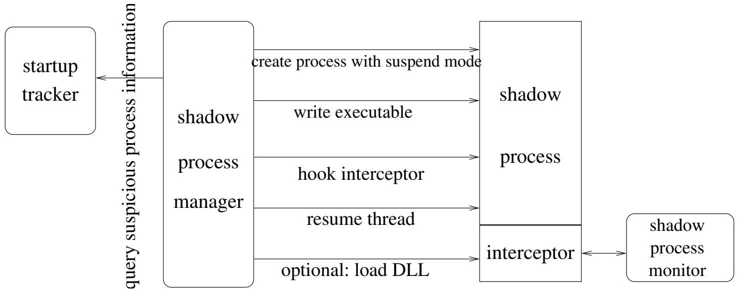


Fig. 3. Shadow Process Manager. A shadow process is started. The corresponding executable could be the dumped image or PE on the disk.

Figure 3 illustrates how `Shadow Process Manager` works to start a shadow process and interfaces with `Startup Tracker` and `Shadow Process Monitor`.

By this way, upon the shadow process startup, `Shadow Process Manager` has already defeated possible usage of shared memory sections in the malware. To deal with the malware anti-detection through process list checking and named objects, `Shadow Process Manager` further intercepts system API calls from the shadow process after the shadow process is started, and

– If the process checks the process list, `Shadow Process Manager` can make the original process invisible to the shadow process. In general, process list checking is through `CreateToolhelp32Snapshot`, which returns a snapshot of all processes, and then `Process32Next` is commonly used to emulate all processes. Thus, `Shadow Process Manager` only needs to intercept `Process32Next`. When `Process32Next` is ready to return the original suspicious process, it is skipped and the next process on the list is returned.
– If the process checks named objects, `Shadow Process Manager` can address this through system API interception. In general, a named object is accessed through a unique name. For example, if a mutex is used, `CreateMutex` takes the name of the mutex as a parameter. `Shadow Process Manager` generates two independent name domains for the original and shadow processes. The interceptor thus can replace the name parameter of the shadow process with a different one to avoid name conflict. The interceptor also returns the original name if the shadow process queries the name of the named object.

Apparently, the above is regarding how to make the original suspicious process invisible to the shadow process. In order to make the shadow process invisible to the original suspicious process, only `Process32Next` needs to be intercepted in the original process.

To facilitate the malware behavior analysis in the `Shadow Process Monitor`, Malyzer also makes the shadow process inaccessible to direct user accesses in order to reduce the analysis noise. For this purpose, Malyzer intercepts `ShowWindow` and `ShowWindowAsync`, always replaces parameter `nCmdShow` with `SW_HIDE`. The shadow process is then inaccessible to direct user input and misbehavior of the shadow process can be accurately captured.

2.2.2 Defeating External Anti-detections: The Shadow Process Behaves Normally to the Outside

Now the shadow process can run in parallel to the original suspicious process. But it may still not behave "correctly" as a normal malware instance.

It is common that malware instances need to interact with some outsider. For example, a bot master controls all bots, and bots running on individual hosts must communicate with the bot master in order to receive commands, send back collected local information, etc. Thus, the shadow process must have the networking capability, and can communicate with outside if needed.

However, in setting up the correct networking functionalities of the shadow process, the outsider, such as a bot master, may have restrictions. For example, multiple connection requests from the same IP address may not be allowed on an IRC server. Thus, it is important that the shadow process should present as another malware instance on a different host to the outsider.

Even without such constraints from the outsider, the malware instance may always use a pre-determined port to communicate with the outside. Once the original process is bound to a particular port, the shadow process cannot use that port, which may cause the malware instance to stop functioning.

Thus, Malyzer must bind the shadow process to a different IP address from the original suspicious process, but use the same port number so that an outsider cannot figure out they are from the same host. That is, Malyzer must be able to support multiple IP addresses.

Usually a process calls `connect` to connect to a remote IP address and calls `bind` to bind to a specific port before it begins to accept connections. The default IP address for these APIs is the primary IP address. In order to bind the shadow process to a secondary IP address, Malyzer needs to intercept these API calls and add an additional IP address in the OS for this purpose.

In Malyzer, when `connect` is called, the socket is always bound to the primary IP address (`A.B.C.D1`) by default. In order to have the shadow process bound to the secondary IP address, Malyzer actively performs binding to (`A.B.C.D2`) before `connect` is called in the API interceptor. When receiving incoming connections, Malyzer intercepts `bind` to bind port to (`A.B.C.D2`) to avoid port conflict. These networking setups guarantee that the shadow process appear to the outsider as a new process running on a different machine.

2.3 Shadow Process Monitor

Malware always performs local sensitive information collection, or controls infected nodes to participate some attacks against a third party. This is fundamental to differentiate a benign process from a malware process. Thus, after the shadow process successfully runs, Malyzer keeps monitoring the behavior of the shadow process in order to determine if it is actually a malware instance.

Considering that ultimately, a malware instance would perform information harvesting or information dispersion, Malyzer concerns three typical kinds of behavior observed from the shadow process:

- **network accesses:** A lot of malware has network activities for various purposes. For example, a trojan sends out the information it collects; a bot contacts its botmaster for commands.
- **hard disk access:** Some malware is designed to collect sensitive information stored on the hard disk of infected hosts. In general, it is normal that a benign process accesses certain directories on the hard disk, such as the current working directory of the process, but it is uncommon to access directories owned by other processes.
- **memory access:** Some malware can directly access memory of other processes for sensitive information (e.g., password) or for further attacks.

In order to decide whether or not a process with a sequence of accesses is malware, existing research commonly applies either anomaly-based analysis or signature-based schemes. But the challenge is that application (including malware) behavior is difficult to predict, considering the complexity of software and diversity of user operations. Fortunately, in Malyzer, the situation is different. The shadow process is a clone of the original suspicious process. Malyzer makes it inaccessible to other processes or users in the host system. This greatly eliminates noisy activities that do not belong to the shadow process (we discuss how to deal with interactive malware with emulated user input later). Without interferences caused by other processes or users on the same host, the behavior of the shadow process could only be autonomous by its inner logic or caused by some remote control. Even though, simply using an anomaly-based approach may not be easy since it is arduous to *manually* define a normal behavior model for various shadow processes of legitimate application processes.

At this end, `Shadow Process Monitor` takes a hybrid approach: to combine both anomaly- and signature-based approaches. Malyzer first defines a set of heuristic malicious behavior rules and then generates the normal process behavior model automatically along with the malware analysis. Malyzer always starts with the anomaly-based approach (the benign process behavior model, consisting of individual process profiles, is empty at the beginning). Malyzer first compares the shadow process behavior with the benign process model. If the model is empty or it cannot make a decision, it is further compared against the heuristic rules. Once the process is determined to be benign, its profile is generated automatically from the captured access sequence and added to the benign process behavior model. Note that an optional component, the user validation, can be added when `Shadow Process Monitor` determines a malware instance. This could improve the accuracy of the analysis. Without such a component, the procedure is fully *automatic.*

Considering three types of accesses, Malyzer defines the heuristic rule set for malicious behavior detection of the shadow process as follows:

- **connection rate (rule I):** When connection count to different destinations in a given time duration is beyond a threshold, an alarm is raised.
- **failed connection rate (rule II):** When the number of failed connections a shadow process makes to different destinations in a given time duration is beyond a threshold, an alarm is raised.

- **command and control channel (rule III):** When the shadow process maintains a certain number of connections beyond a threshold or when it maintains a connection to a destination for a duration beyond a threshold, an alarm is raised.
- **sensitive file on disk (rule IV):** When the shadow process accesses directories other than system directory and current working directory, an alarm is raised.
- **sensitive data in memory (rule V):** When the shadow process reads or writes the memory of other processes, an alarm is raised.

Combination of these heuristic rules could be applied too. Note that these heuristic rules are effective in Malyzer because a significant amount of noise caused by users has been eliminated.

On the other hand, for each process, the normal behavior model consists of the process profile list, a list of relevant API calls and corresponding parameters when network, disk, and memory accesses are invoked in the shadow process.

3 Malyzer Implementation

To demonstrate the concept and for experiments, we implement a prototype of Malyzer on Windows XP Professional. As we have shown in Figure 3, the three major components of Malyzer interact with each other. Among them, `Startup Tracker` is implemented as a driver to keep track of process creation. It keeps monitoring and intercepting system daemons, such as `services.exe`, `svchost.exe`, `lsass.exe`, `spoolsv.exe`, and `system.exe`. This is done through directly replacing their import tables entries [26]. In addition, `Startup Tracker` keeps intercepting process creation related APIs, such as `createProcess`, `writeProcessMemory`, and `resumeThread`, which is done through Microsoft `Detours 2.1 Express` [4]. Upon a child process is created by an intercepted process, relevant APIs in child process are also intercepted in a recursive fashion.

If `Startup Tracker` detects dynamic code replacement via the unique API call sequence as we have discussed in section 2, `Startup Tracker` will dump the initial memory of the process. Memory dumping is conducted before the process is actually resumed. In addition, the EAX register containing the entry point address, obtained from the parameter of `resumeThread`, is kept. `Startup Tracker` stores this information for `Shadow Process Manager`.

`Shadow Process Manager` is responsible for starting a shadow process on demand, given a suspicious process to analyze. `Shadow Process Manager` first queries the module file name of the suspicious process and creates a process with suspended mode from the corresponding executable.

By querying `Startup Tracker`, if the suspicious process is not started via dynamic code replacement, `Shadow Process Manager` reads the executable again, aligns, and copies that to the suspended process again (in order to defeat direct memory sharing in malware instances). This procedure is the same as dynamic code replacement. In addition, `Shadow Process Manager` also calculates the entry point address and sets EAX for the shadow process [18].

If the suspicious process is started via dynamic code replacement, Shadow Process Manager reads the memory image dumped by Startup Tracker and copies to the suspended process. As Startup Tracker also keeps the EAX value of the suspicious process, Shadow Process Manager can calculate the entry point from the original EAX value and set accordingly in the shadow process.

Before resuming the shadow process, Shadow Process Manager hooks interceptors to the shadow process in order to detect various anti-detection approaches that could be used by a malware instance and monitor its behavior.

Some of the intercepted API calls and their parameters from Shadow Process Manager are also fed into Shadow Process Monitor for further decisions. These are disk, network, and memory accessing APIs and their parameters. Such a call is an event to trigger Shadow Process Monitor. In the current implementation, Shadow Process Monitor maintains a cache, which contains shadow process profiles. Once an API calling event is fed to Shadow Process Monitor, it compares with its cached profiles. Currently, the API name and the corresponding parameters are compared. A discrepancy will raise an alarm.

For heuristic rules, we have set up thresholds as follows: the connection rate and the failed connection rate are set as ten and five per minute. For the command and control channel, the connection duration threshold for a persistent channel is 30 seconds [7]. If the shadow process is finally determined to be benign, its API calls are recorded and updated in the cache.

In addition to capture behavior of malware shadow process, Malyzer also always locates the correct malcode source. The difficulty lies in that if the malware runs via DLL injection, such as BHO, Malyzer is expected to report which DLL the API caller is from. In our current implementation, Malyzer queries stack information [13] to find the caller upon an API call triggering an alarm. For this purpose, Malyzer defines a macro that reads register EBP, which contains the value of frame pointer of the caller. Subsequently, the return address is stored at (EBP + 4). With the return address, Malyzer can query the module where the return address resides.

4 Malyzer Evaluation

With the prototype, we test Malyzer against a number of malware samples that use different anti-detection mechanisms. Some representative experimental samples that use different anti-detections are listed in Table 1. We experiment on all malware samples. Due to page limit, we will only present some interesting ones with different anti-detection mechanisms.

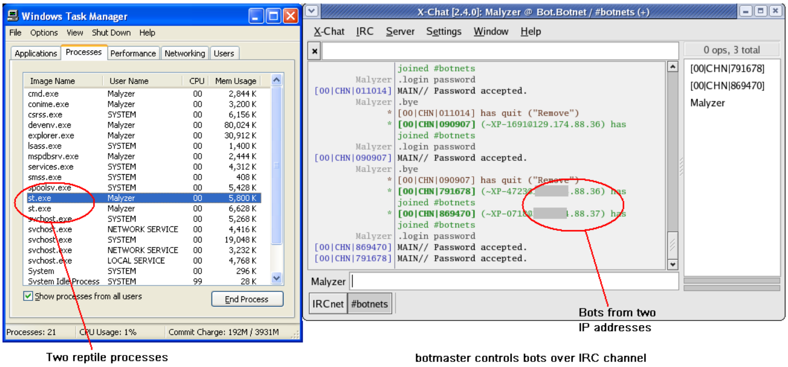
4.1 Whether Malyzer Can Defeat Malware Anti-detections

We first test whether a shadow process of them can be successfully started on the same host where there is already one malware instance running.

reptile is the most interesting sample we tested. In this version, reptile performs various environment tests to defeat virtual machines and various

Table 1. Malware Samples and their Anti-detections

malware	anti-detection actions
agobot3	check process list for processes with the same name perform VMware detection
forBot	check process list for processes with the same name
graybird	start with dynamic code replacement
Gorgon trojan	check process list for debugger process OLLYDBG.EXE
JrBot	use mutex to prevent multiple copies from running on a host
reptile	perform debugger, VMware, SoftIce detection perform BreakPoint, Single Step detection use mutex to prevent multiple copies from running on a host
rBot	use mutex to prevent multiple copies from running on a host
sdbot05	check process list for processes with the same name
spybot	perform VMware detection
storm worm	perform VMware detection
trojan downloader	Search for Wireshark, ZoneAlarm, Olly Debug



Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users

Image Name	User Name	CPU	Mem Usage
cmd.exe	Malyzer	00	2,844 K
conime.exe	Malyzer	00	3,200 K
csrss.exe	SYSTEM	00	6,156 K
devenv.exe	Malyzer	00	80,024 K
explorer.exe	Malyzer	00	30,912 K
lsass.exe	SYSTEM	00	1,400 K
mspdbsrv.exe	Malyzer	00	2,444 K
services.exe	SYSTEM	00	4,312 K
smss.exe	SYSTEM	00	408 K
spoolsv.exe	SYSTEM	00	5,428 K
st.exe	Malyzer	00	5,800 K
st.exe	Malyzer	00	6,628 K
svchost.exe	SYSTEM	00	5,268 K
svchost.exe	NETWORK SERVICE	00	4,416 K
svchost.exe	SYSTEM	00	19,048 K
svchost.exe	NETWORK SERVICE	00	3,232 K
svchost.exe	LOCAL SERVICE	00	4,768 K
System	SYSTEM	00	296 K
System Idle Process	SYSTEM	99	28 K

☑ Show processes from all users

Processes: 21 CPU Usage: 1% Commit Charge: 192M / 3931M

Two reptile processes

X-Chat [2.4.0]: Malyzer @ Bot.Botnet / #botnets (+)
X-Chat IRC Server Settings Window Help
0 ops, 3 total

```
                 joined #botnets
         Malyzer .login password
[00|CHN|011014] MAIN// Password accepted.
         Malyzer .bye
               * [00|CHN|011014] has quit ("Remove")
               * [00|CHN|090907] (~XP-1691@129.174.88.36) has
                 joined #botnets
         Malyzer .login password
[00|CHN|090907] MAIN// Password accepted.
         Malyzer .bye
               * [00|CHN|090907] has quit ("Remove")
               * [00|CHN|791678] (~XP-4723@[redacted].88.36) has
                 joined #botnets
               * [00|CHN|869470] (~XP-0718@[redacted].88.37) has
                 joined #botnets
         Malyzer .login password
[00|CHN|869470] MAIN// Password accepted.
[00|CHN|791678] MAIN// Password accepted.
```

[00|CHN|791678]
[00|CHN|869470]
Malyzer

Bots from two IP addresses

botmaster controls bots over IRC channel

Fig. 4. A shadow process of `reptile` is started and running on the same host

debuggers and debugging techniques. It also prevents multiple copies from running on the same host via mutex. When `reptile.exe` starts, it copies itself to `%windir%\st.exe` and launches `st.exe` before it exits. Subsequently, `st.exe` deletes `reptile.exe` on the disk. After `Shadow Process Manager` starts a shadow process of this malware instance, Figure 4 shows that two processes are running successfully within Malyzer. `Graybird` runs via dynamic code replacement so that it can disguise as an IE process. `Startup Tracker` detects this approach and dumps the initial process memory. Accordingly, `Shadow Process Manager` starts a shadow process successfully. Figure 5 shows that two graybird processes are running in the context of two IE processes. `agobot3` is a very common bot that employs the process list checking to prevent multiple copies from running on the same machine. Figure 6 shows that in Malyzer, a shadow process of `agobot3` can be started and run in parallel with the original one successfully.

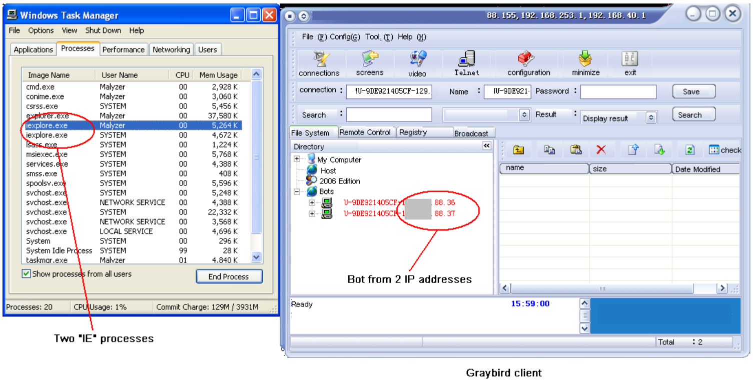

Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users

Image Name	User Name	CPU	Mem Usage
cmd.exe	Malyzer	00	2,928 K
conime.exe	Malyzer	00	3,060 K
csrss.exe	SYSTEM	00	5,456 K
explorer.exe	Malyzer	00	37,580 K
iexplore.exe	Malyzer	00	5,264 K
iexplore.exe	SYSTEM	00	4,672 K
lsass.exe	SYSTEM	00	1,224 K
msiexec.exe	SYSTEM	00	5,768 K
services.exe	SYSTEM	00	4,388 K
smss.exe	SYSTEM	00	408 K
spoolsv.exe	SYSTEM	00	5,596 K
svchost.exe	SYSTEM	00	5,248 K
svchost.exe	NETWORK SERVICE	00	4,388 K
svchost.exe	SYSTEM	00	22,332 K
svchost.exe	NETWORK SERVICE	00	3,568 K
svchost.exe	LOCAL SERVICE	00	4,696 K
System	SYSTEM	00	296 K
System Idle Process	SYSTEM	99	28 K
taskmgr.exe	Malyzer	01	4,840 K

☑ Show processes from all users

Processes: 20 CPU Usage: 1% Commit Charge: 129M / 3931M

Two "IE" processes

connection : 1U-9DE921405CF-129. Name : 1U-9DE921· Password :

Search : Result : Display result

My Computer
Host
2006 Edition
Bots
U-9DE921405CF-1[redacted].88. 36
U-9DE921405CF-1[redacted].88. 37

Bot from 2 IP addresses

Ready 15:59:00

Total : 2

Graybird client

Fig. 5. A shadow process of `graybird` is started and running on the same host. `graybird` starts itself via dynamic code replacement.

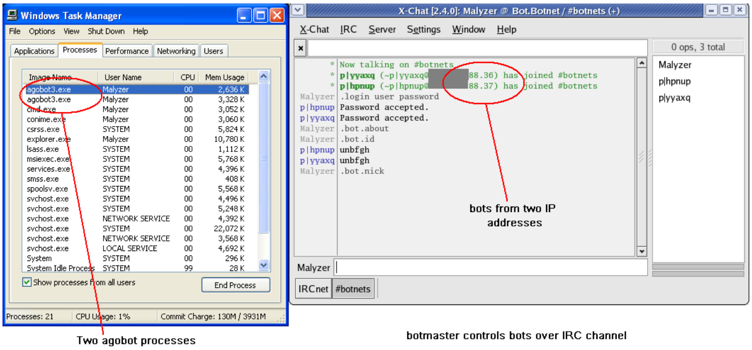

Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users

Image Name	User Name	CPU	Mem Usage
agobot3.exe	Malyzer	00	2,636 K
agobot3.exe	Malyzer	00	3,328 K
cmd.exe	Malyzer	00	3,052 K
conime.exe	Malyzer	00	3,060 K
csrss.exe	SYSTEM	00	5,824 K
explorer.exe	Malyzer	00	10,780 K
lsass.exe	SYSTEM	00	1,112 K
msiexec.exe	SYSTEM	00	5,768 K
services.exe	SYSTEM	00	4,396 K
smss.exe	SYSTEM	00	408 K
spoolsv.exe	SYSTEM	00	5,568 K
svchost.exe	SYSTEM	00	4,496 K
svchost.exe	SYSTEM	00	5,248 K
svchost.exe	NETWORK SERVICE	00	4,392 K
svchost.exe	SYSTEM	00	22,072 K
svchost.exe	NETWORK SERVICE	00	3,568 K
svchost.exe	LOCAL SERVICE	00	4,692 K
System	SYSTEM	00	296 K
System Idle Process	SYSTEM	99	28 K

☑ Show processes from all users

Processes: 21 CPU Usage: 1% Commit Charge: 130M / 3931M

Two agobot processes

X-Chat [2.4.0]: Malyzer @ Bot.Botnet / #botnets (+)
X-Chat IRC Server Settings Window Help
IRCnet
#botnets

```
      * Now talking on #botnets
      * p|yyaxq (~p|yyaxq@[redacted]88.36) has joined #botnets
      * p|hpnup (~p|hpnup@[redacted]88.37) has joined #botnets
Malyzer .login user password
p|hpnup Password accepted.
p|yyaxq Password accepted.
Malyzer .bot.about
Malyzer .bot.id
p|hpnup unbfgh
p|yyaxq unbfgh
Malyzer .bot.nick
```

0 ops, 3 total

Malyzer
p|hpnup
p|yyaxq

bots from two IP addresses

botmaster controls bots over IRC channel

Fig. 6. A shadow process of `agobot3` is started and running on the same host. `agobot3` simply checks the process list to prevent duplicated copies from the same IP address.

In these tests, both `agobot` and `reptile` are IRC-based malware. They need to connect to an IRC server to receive commands. By default, the IRC server only accepts one connection per IP address. Because Malyzer binds the original process and the shadow process to a different IP address, although they are running on the same host, the IRC server takes them as two different instances running on two different machines, as shown in the above figures.

4.2 Whether a Shadow Process Functions Normally and Whether Its Misbehavior Can Be Detected

The previous tests show that the shadow process of a malware sample can be successfully started. However, whether the shadow process can still function "properly" is not clear, which is critical for the further detection and analysis.

Table 2. Detection of Information Harvesting and Dispersion by rBot

action	event trigger	detection rule
After the process starts, it connects to the bot master.	`connect`: shadow process keeps a consistent connection	rule III: command and control channel
After rBot receives a `advscan` command, it starts random port scan.	`connect`: shadow process sends out packets to different destinations, the connection rate is beyond a threshold.	rule I and rule II: connection rate and failed connection rate
After rBot receives a `getcdkeys` command, it retrieves cdkey file.	`ReadFile`: shadow process accesses file of other applications.	rule IV: sensitive data access on disk

To verify that a shadow process can work "correctly" as if it is truly running on a different host, we experiment by instructing the shadow process for various attacks. Along with these experiments, we can also test whether their misbehavior can be accurately captured so that a decision is made through `Shadow Process Monitor`. At the beginning, the cache in `Shadow Process Monitor` is empty, and the heuristic rules are used. Table 2 shows the results when `rBot` is instructed to perform local and remote attacks. For local attacks, a command of `getcdkeys` from our IRC server is sent to the bot to search for product CD keys. For remote attacks, a command of `advscan` is sent to instruct the bot to perform random port scan. In the experiments, all malware actions are correctly captured by Malyzer.

5 Malyzer Optimization and Further Discussion

In this section, we discuss some issues with the current design and implementation of Malyzer as well as possible improvement.

First, our experiments tested automatic malware. For malware whose misbehavior is triggered by certain user activities, such as accessing a specific Web site, the current Malyzer implementation has not included a component to use emulated user input to allure malware actions. We are adopting the approach taken by Panorama [31] to emulate the user input to trigger the malware.

Second, in the design space, `Shadow Process Monitor` aims to capture misbehavior of shadow processes. This approach is similar to existing behavior-based approaches with the understanding that information harvesting and information dispersion are the essential behaviors differentiating malware processes from benign ones. Other approaches and systems (e.g., Snort, Bro) could be integrated with this component. In addition, the current cache implementation of `Shadow Process Monitor` is rather simple for the demonstration of concept. We would like to define a general and portable format so that once a shadow process profile is created, it can be shared and distributed.

Third, the current implementation of Malyzer relies on behavior analysis through API call interceptors. A malware developer could evade Malyzer without calling such APIs by coding its own functions or calling some system native functions that we are not aware of to evade Malyzer. Although Malyzer can be implemented at the native system call level to defeat these efforts, fundamentally, a malware developer can defeat Malyzer by implementing in assembly code or using timing correlation to detect the existence of Malyzer by looking into the time duration of calling various APIs. Although our usage of Detours causes trivial processing overhead as reflected by our successful experiments with `reptitle`, which uses API timing to defeat "Single Step", more precise timing could enforce Malyzer to intercept more APIs in order to defeat such efforts.

On the other hand, as a malware analyzer, Malyzer works under the assumption that the host is not subverted through some rootkits. Whether at the stage to unveil malware camouflaging or de-activate various malware anti-detection mechanisms, Malyzer needs a trustworthy underlying operating system. Once the underlying operating system is completely subverted, the information Malyzer obtains could be wrong, which would lead to analysis failure.

In addition, for duplicated process checking, a malware instance could mark the registry or a file so that it can query the mark at the startup. While Malyzer cannot defeat this, such an anti-detection approach is not reliable because if exceptions happen (e.g., powering down), the malware cannot restart.

Lastly, today a lot of malware packers are available for malcode polymorphism, obfuscation, encryption, and some anti-detection as well. For example, Themida [5] provides anti-debug, anti-tracing, anti-dumping and VM detection options. While source polymorphism, obfuscation, and encryption do not affect Malyzer, since Malyzer uses Detours to intercept API calls, which intercepts Win32 functions by re-writing target function code in memory, some packers can use IAT destruction to completely change the PE structure of malcode. In this case, Detours fails to intercept API calls. As an alternative, proxy DLL [6] does not rely on the IAT structure which is a good candidate of the API interceptor. Malyzer thus can take this approach.

6 Conclusion

Various anti-detection techniques have been practically employed by malware developers to create more and more stealthy Internet malware. The success of malware analysis and detection heavily depends on whether malware analysts can defeat all kinds of malware anti-detection mechanisms. In this paper, we have made an initial step towards effectively unveiling various malware camouflaging at startup and runtime through the design and implementation of Malyzer. Through various countering anti-detection measures, Malyzer can accurately capture malware behavior. Experiments have been conducted to evaluate Malyzer with various malware samples. The results demonstrate the effectiveness of Malyzer.

Acknowledgment

We thank the anonymous referees for providing constructive comments. The work has been supported in part by U.S. AFOSR under grant FA9550-09-1-0071, and by U.S. National Science Foundation under grants CNS-0509061, CNS-0621631, and CNS-0746649.

References

1. http://www.pctools.com/mrc/infections/id/Webbuying/
2. http://news.softpedia.com/newsTag/Graybird
3. http://blogs.windowsecurity.com/parker/2006/07/11/malware-packers/
4. http://research.microsoft.com/sn/detours/
5. http://www.oreans.com/ThemidaWhatsNew.php
6. http://www.codeproject.com/KB/system/hooksys.aspx
7. Taxonomy of botnet threats (November 2006), http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/botnettaxonomywhitepapernovember2006.pdf
8. Brumley, D., Newsome, J., Song, D., Wang, H., Jha, S.: Towards automatic generation of vulnerability-based signatures. In: Proceedings of IEEE Symposium on Security and Privacy, Berkely/Oakland, CA (May 2006)
9. Butler, J., Hoglund, G.: Vice-catch the hookers! (July 2004)
10. Chiang, K., Lloyd, L.: A case study of the rustock rootkit and spam bot. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA (April 2007)
11. Chow, J., Pfaff, B., Garfinkel, T., Christopher, K., Rosenblum, M.: Understanding data lifetime via whole system simulation. In: Proceedings of the 13th USENIX Security Symposium (August 2004)
12. Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., Barham, P.: Vigilante: End-to-end containment of internet worms. In: Proceedings of SOSP, Brighton, United Kingdom (October 2005)
13. Dimitrov, C.: Playing with the stack, http://www.codeproject.com/tips/stackdumper.asp
14. Desclaux Fabrice. Skype uncovered, http://www.ossir.org/windows/supports/2005/2005-11-07/EADS-CCR_Fabrice_Skype.pdf
15. Grizzard, J., Sharma, V., Nunnery, C., Kang, B., Dagon, D.: Peer-to-peer botnets: Overview and case study. In: Proceedings of the HotBots, Cambridge, MA (April 2007)
16. Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic. In: Proceedings of the 15th NDSS, San Diego, CA (February 2008)
17. Kang, M., Poosankam, P., Yin, H.: Renovo: A hidden code extractor for packed executables. In: Proceedings of WORM, Alexandria, VA (November 2007)
18. Keong, T.: Dynamic forking of win32 exe, http://www.security.org.sg/code/loadexe.html
19. Kim, H., Karp, B.: Autograph: Toward automated distributed worm signature detection. In: Proceedings of USENIX Security, San Diego, CA (August 2004)
20. Li, Z., Sanghi, M., Chen, Y., Kao, M., Chavez, B.: Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience. In: Proceedings of IEEE Symposium on Security and Privacy, Berkely/Oakland, CA (May 2006)

21. Liu, L., Chen, S., Yan, G., Zhang, Z.: Bottracer: Execution-based bot-like malware detection. In: Proceedings of the 11th Information Security Conference, Taipei, China (September 2008)
22. Moshchuk, A., Bragin, T., Deville, D., Gribble, S., Levy, H.: Spyproxy: Execution-based detection of malicious web content. In: Proceedings of the 16th USENIX Security Symposium, Boston, MA (August 2007)
23. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically generating signatures for polymorphic worms. In: Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA (May 2005)
24. Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proceedings of the 12th NDSS (February 2005)
25. Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N.: The ghost in the browser analysis of web-based malware. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA (April 2007)
26. Richter, J.: Programming applications for microsoft windows
27. Rutkowaska, J.: System virginity verifier: Defining the roadmap for malware detection on windows systems (September 2005)
28. Singh, S., Estan, C., Varghese, G., Savage, S.: Automated worm fingerprinting. In: Proceedings of OSDI, San Francisco, CA (2004)
29. Stinson, E., Mitchell, J.C.: Characterizing the remote control behavior of bots. In: Hämmerli, B.M., Sommer, R. (eds.) DIMVA 2007. LNCS, vol. 4579, pp. 89–108. Springer, Heidelberg (2007)
30. Wang, Y., Roussev, R., Verbowski, C., Johnson, A., Wu, M., Huang, Y., Kuo, S.: Gatekeeper: Monitoring auto-start extensibility points (aseps) for spyware management. In: Proceedings of LISA (November 2004)
31. Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E.: Panorama: Capturing system-wide information flow for malware detection and analysis. In: Proceedings of ACM CCS, Alexandria, VA (October 2007)

A New Message Recognition Protocol with Self-recoverability for Ad Hoc Pervasive Networks

Ian Goldberg[1], Atefeh Mashatan[2], and Douglas R. Stinson[1]

[1] David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, Ontario Canada N2L 3G1
http://crysp.uwaterloo.ca/

[2] The Security and Cryptography Laboratory (LASEC), EPFL
CH-1015 Lausanne, Switzerland
http://lasecwww.epfl.ch/

Abstract. We examine the problem of message recognition by reviewing the definitions and the security model in the literature. In particular, we examine the Jane Doe protocol, which was proposed by Lucks et al., more closely and note its inability to recover in case of a certain adversarial disruption. Our paper saves this well-studied protocol from its unrecoverable state when such adversarial disruption occurs. We propose a new message recognition protocol, which is based on the Jane Doe protocol, and incorporate the resynchronization technique within the protocol itself. That is, without having to provide a separate resynchronization procedure, we overcome the recoverability problem of the Jane Doe protocol. Moreover, we enumerate all possible attacks against the new protocol and show that none of the attacks can occur. We further prove the security of the new protocol and its ability to self-recover once the disruption has stopped.

Keywords: Cryptographic Protocols, Authentication, Recognition, Self-Recoverability, Pervasive Networks, Ad Hoc Networks.

1 Introduction

Entity recognition is a weaker security notion than entity authentication; it refers to the process where two parties meet initially and one party can be assured in future conversations that it is communicating with the same second party. There is an analogous correspondence between message recognition and message authentication.

There have been several recent papers on designing protocols where the source of trust is a narrow-band authenticated channel; see for example [4], [7], [9], [10], and [11]. In particular, there has been recent interest in designing recognition protocols using this communication model. This problem has been considered in a context where we are dealing with low-computational power devices which cannot handle public-key computations and where no pre-deployed shared secret

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 219–237, 2009.

exists. On the other hand, the devices have access to a narrow-band authenticated channel at the initialization step and are later placed in a constrained, possibly hostile, insecure environment.

Lucks et al. [4] motivated this model with the following example. Let Alice and Bob be two strangers who meet in a party for the first time. They leave the party after making a bet. Some days later, it turns out that Alice wins the bet. Afterward, Bob receives a message claiming to be sent from Alice. The message includes a bank account number and asks Bob to deposit Alice's prize to that bank account. Bob wants to be assured that this message is indeed sent from the entity who introduced herself as "Alice" in the party. In other words, Bob needs to *recognize* "Alice", whoever she is, or a message that is sent from her.

Now consider Alice and Bob to be two small devices who "meet" in a somewhat secure environment that allows them to send authenticated, but not confidential, messages. They are later placed in a hostile environment where Alice wants Bob to recognize the messages sent from her to Bob. An adversary, Eve, is present all along. When Alice and Bob first meet, Eve can read the authenticated messages, but cannot change them. Later, when Alice and Bob are placed in a hostile environment, Eve can not only read, but also modify messages. She can also insert her own messages claiming to be from either party. Eve's goal is to make Bob accept messages from her as sent from Alice, where Alice has never, or at least not recently, sent those messages.

Since message recognition is weaker than message authentication, every message authentication protocol trivially provides message recognition. Moreover, message recognition can be achieved using public-key, when public-key computations are feasible, or secret-key cryptography, when pre-deployed authentic information is available. However, in some scenarios, public-key computations may be too costly and there may be no secure channel where the secret keys can be transmitted confidentially.

One can ask what security goals can be achieved in such a constrained model? There are claims in the literature, see [11] for example, suggesting that achieving message authentication is not possible in such an environment. Hence, they pursue the weaker security of message recognition.

We examine the Jane Doe message recognition protocol proposed by Lucks et al. in more detail and note that in case of a particular adversarial disruption, this protocol fails to recover. In other words, the adversary can trap one party in a state that he or she will no longer accept legitimate messages that were sent by the other party. This inability to recover was noted previously in [8], where it was fixed by calling upon a separate procedure called a "resynchronization protocol". Here, we propose a new message recognition protocol that is able to recover without having to call a separate resynchronization protocol. That is, our new protocol has the advantage of self-recoverability. (The fact that self-recoverability is built into our protocol means that the parties involved do not have to negotiate when to resynchronize. This makes the whole system simpler and more robust.) We also formally prove that our new protocol is secure and fully recovers once the disruptions have stopped.

The rest of the paper is organized as follows. Section 2 is devoted to examining previous recognition protocols and noting their shortcomings. In Section 3, we describe a new message recognition protocol. Finally, Section 4 is devoted to proving the security and recoverability of the protocol.

2 Previous Recognition Protocols

In this section, we briefly review the existing message recognition protocols and discuss their usability in the context of networks with low-computational power devices that also have low communication bandwidth.

There are two communication channels considered in the setting of recognition protocols: an insecure broadband channel, denoted by $\rightarrow$, and an authenticated non-confidential narrow-band channel denoted by $\Rightarrow$. The broadband channel is available all the time and the narrow-band channel is only accessible once, for the initial session between two users.

The Guy Fawkes protocol was proposed by Anderson et al. [1]. There are two variants of this protocol suggested and a one-way hash function is deployed in both variants. In the first variant, random codewords are chosen in each session and are refreshed each time a message is authenticated. Alice commits to the message and the codewords and then publishes the commitment in a public directory which provides time-stamping services. Later, she reveals the committed values to prove that she is the same party who was involved in previous sessions. However, assuming the existence of a trusted party which provides time-stamping services is not realistic in most ad hoc network scenarios. The second variant does not require any interaction with a time-stamping provider and instead requires interaction of the authenticating party with the verifying party. The initialization phase of this protocol does not assume any authenticated channel; however, it requires digital signatures for authenticating the first blocks and codewords. This may not be suitable in ad hoc networks and, in particular, in low-power environments. Moreover, for a message to be authenticated in session i, users need to commit to it in the previous session. In the context of message recognition, this means that users are engaged in two sessions of this protocol to authenticate a single message, which may not be desirable.

The Remote User Authentication Protocol is an entity recognition protocol that was introduced by Mitchell [9]. In this protocol, a message authentication code (MAC) is used to prove that a user is the same entity involved in previous sessions. The protocol can be adapted to perform message recognition as well; however, this is not discussed in the paper. The setup phase of this protocol requires that t MAC values be sent over the authenticated channel. This may be costly since authenticated channels are usually of low bandwidth. Further, the “cut-and-choose” procedure in each round involves sending $2t$ MAC values and r secret keys. In order for the protocol to be secure, it is suggested that $t \geq 35$ and $r \approx t/2$. Hence, the amount of computation and communication here is large compared to other protocols that are providing entity or message recognition and it may not be suitable for settings with low-power devices.

Weimerskirch et al. [11] proposed a protocol called Zero Common-Knowledge (ZCK). This protocol is the starting point of a series of recent publications; see for example [3], [4], [5], [8], [6]. The ZCK protocol uses message authentication codes (MACs) and hash chains of the form $a_i = H(a_{i-1})$ and $b_i = H(b_{i-1})$, $i = 1, \dots, n$, as keys for the MACs. The length of the hash chain, n, is fixed at the beginning and H is a one-way hash function.

Hammell et al. [3] implemented the ZCK protocol and provided measurements and observations as a proof-of-concept. They investigated whether the ZCK protocol suits devices with low computational power, low code space, low communication bandwidth, low energy resources. They concluded that it does exhibit these requirements, however, denial-of-service and memory complexity are areas of concern and needed to be addressed or improved upon in the future.

Hammell et al. did not investigate the security properties of the ZCK protocol, but rather relied on the security proof that came along with it. However, Lucks et al. [4] found a mistake in the security proof of this protocol and presented a practical attack against it. Moreover, using the same idea of using values in a hash chain as keys for MACs, they proposed a message recognition protocol that guards against the found attack. We describe the protocol proposed by Lucks et al. in more detail; it has been named the Jane Doe protocol [5].

A one-way hash function $H : \{0,1\}^s \rightarrow \{0,1\}^s$ and a message authentication code MAC : $\{0,1\}^s \times \{0,1\}^* \rightarrow \{0,1\}^c$ are considered as building blocks of this protocol. Typical parameters are suggested to be $s \geq 80$ and $c \geq 30$. The maximum number of messages to be authenticated, or the maximum number of sessions, in the Jane Doe protocol is fixed to be n. Alice randomly chooses a_0 and forms a hash chain of the form $a_i = H(a_{i-1})$, $i = 1, \dots, n$. Similarly, Bob randomly chooses b_0 and forms $b_i = H(b_{i-1})$, $i = 1, \dots, n$. Alice and Bob will respectively use a_i and b_i as keys for MAC values they compute in session i.

The initialization phase is constituted of Alice and Bob exchanging the values of a_n and b_n. In this phase of the execution, Eve is passive and the communication is denoted by $\Rightarrow$.

There will be n sessions of the protocol and we denote them in descending order by $n-1, \dots, 0$; this is because the values of the hash chains are going to be revealed in this order. In each session i, Alice would like to authenticate a message m_i. She uses a_i as the key for the MAC and sends the MAC value of m_i to Bob. Bob then authenticates himself to Alice by revealing b_i. Once Alice has verified b_i, she reveals a_i. Then a_i allows Bob to verify Alice and m_i. Once the session is over, Alice and Bob "move down" in the hash chain and use a_{i-1} and b_{i-1} as keys for session $i-1$.

Lucks et al. write accept-key(k) when a key k has been accepted, and commit-message(m, i) when Alice commits herself to authenticate m in session i. Similarly, accept-message(m, i) indicates that Bob has accepted m as sent from Alice in session i. The formal description of the Jane Doe protocol is given next.

Alice's internal state in the Jane Doe protocol is as follows:

- i, the session counter
- b_{i+1}, the most recently accepted value of Bob's hash chain (hence accept-key(b_{i+1}) has occurred already)
- a one-bit flag, to distinguish the program states **A0** and **A1**.

Similarly, Bob's internal state is:

- i, the session counter
- a_{i+1}, the most recently accepted value of Alice's hash chain (hence accept-key(a_{i+1}) has occurred already)
- a one-bit flag, to distinguish the program states **B0** and **B1**.

Session i of the Jane Doe protocol:

```
A0 (Alice's initial program state) Obtain $m_i$ (possibly from Eve), then
   Commit-message($m_i, i$).
   Compute $d_i = \mathrm{MAC}_{a_i}(m_i)$.
   Send $(d_i, m_i)$; goto A1.
A1 Wait for a message $b'$ (supposedly from Bob), then
   If $H(b') = b_{i+1}$ then
   Let $b_i := b'$, accept-key($b_i$) and send $a_i$. Let $i := i - 1$ and goto A0
   else goto A1.
B0 (Bob's initial program state) Wait for a message $(d_i, m_i)$, then send $b_i$ and
   goto B1.
B1 Wait for a message $a'$ (supposedly from Alice), then
   If $H(a') = a_{i+1}$ then
   Let $a_i := a'$ and accept-key($a_i$).
   If $\mathrm{MAC}_{a'}(m_i) = d_i$ then
   Accept $m_i$ as authentic in session $i$
   (else do not accept any message for session $i$).
   Let $i := i - 1$ and goto B0
   else goto B1.
```

Lucks et al. present the Jane Doe protocol in an extended abstract [4], and prove its security in the full version of the paper [5]. The Jane Doe protocol is proved to be secure given that the preimage resistance, second preimage resistance, and unforgeability properties, and their hash chain equivalents, hold. These properties are described in Section 4.

Although the Jane Doe protocol is provably secure, it nonetheless falls short in case of a certain adversarial disruption. In particular, Eve can easily manipulate one party to move forward to the next session, while the other party is still in the previous session. In such a case, a party could get trapped in a state and never be able to finish execution of a session; as a result, he or she remains stuck in that state forever.

Figure 1 illustrates a situation where Bob is trapped by Eve in program state **B1**. The condition in program state **B1** fails since $a_{i+1} \neq H(a'_i)$. This will cause Bob to stay in **B1** waiting for a new a_i. Now even if Alice sends him a legitimate message m_i, he will ignore it. Although this looks like a denial of service attack, it is much stronger than that. Eve can go away and yet Alice and Bob are still unable to communicate because Bob is trapped. The details of the disruption are as follows.

Eve sends m'_i and d'_i to Bob and he will automatically decrement his index to i while Alice does not. Eve chooses a'_i such that $a_{i+1} \neq H(a'_i)$, which will make Bob wait for a new a_i. While he is waiting for a new a_i, he will not accept a message of the form (m_j, d_j), for any j. Hence, even if Alice sends him a legitimate message, he will ignore it. As a result, he is "trapped" in state **B1**.

Lucks et al. suggest that Bob sends b_i again after he has waited for too long to receive the correct a_i. However, when Alice has not initiated the session and is not anticipating b_i, it is not clear what she is supposed to do. Hence, this will not help the protocol recover in case of this particular disruption.

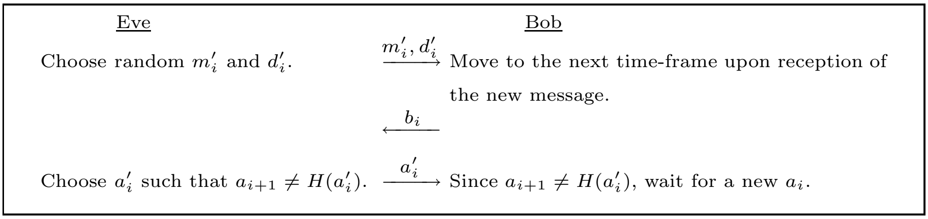

Eve
Bob
Choose random m'_i and d'_i.
$\xrightarrow{m'_i, d'_i}$
Move to the next time-frame upon reception of the new message.
$\xleftarrow{b_i}$
Choose a'_i such that $a_{i+1} \neq H(a'_i)$.
$\xrightarrow{a'_i}$
Since $a_{i+1} \neq H(a'_i)$, wait for a new a_i.

Fig. 1. Eve "trapping" Bob in state **B1**

Eve can play the same trick with Alice and trap her in program state **A1** for an indeterminate period of time; Figure 2 illustrates this situation.

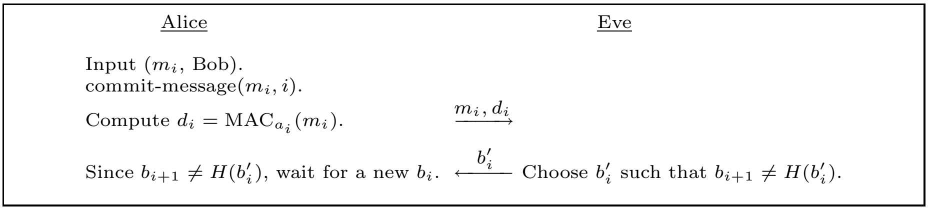

Alice
Eve
Input $(m_i$, Bob).
commit-message(m_i, i).
Compute $d_i = \mathrm{MAC}_{a_i}(m_i)$.
$\xrightarrow{m_i, d_i}$
Since $b_{i+1} \neq H(b'_i)$, wait for a new b_i.
$\xleftarrow{b'_i}$
Choose b'_i such that $b_{i+1} \neq H(b'_i)$.

Fig. 2. Eve "trapping" Alice in state **A1**

Once again, we note that this inability to recover is a problem since the adversary does not need to continue her active involvement. She can leave the network and yet Alice and Bob will no longer be able to have successful communication. This renders the protocol unusable in practice.

In the next section, we propose a message recognition protocol which attains self-recoverability in case of the noted disruptions. It is in fact a highly nontrivial task to modify the protocol to achieve self-recoverability. Because the entities may be in additional "states", depending on the information they possess and its authenticity, the protocol is necessarily more complicated. As a consequence, the security proof is more difficult.

3 A New Message Recognition Protocol

We describe the details of our proposed recognition protocol in this section, while the security and recoverability analyses are postponed to the next session. Although this protocol is based on the Jane Doe protocol proposed by Lucks et al., the logic of the instructions of Alice and Bob has changed considerably. Moreover, the information exchanged between Alice and Bob has changed as well.

Note that each pair of users can execute this new protocol. However, as in the Jane Doe protocol, there must be a different pair of hash chains for each pair of communicating users. It is implicitly assumed that Alice and Bob are the communicating parties in the rest of the paper.

The initialization phase and the setup of the hash chains are exactly as in the Jane Doe protocol. The internal state of Alice includes (along with each variable's initial value):

- $i_A := n - 1$: the position of Alice in her chain.
- $i_{acceptA} := n$: the last index of Bob's chain that was accepted by Alice.
- $b_A := b_n$: the last value of Bob's chain that was accepted by Alice.
- $M := Null$: the input message to be authenticated in the current session.
- a one-bit flag, to distinguish the program states **A0** and **A1**.

Similarly, Bob's internal state is as follows:

- $i_B := n - 1$: the position of Bob in his chain.
- $i_{acceptB} := n$: the last index of Alice's chain that was accepted by Bob.
- $a_B := a_n$: the last value of Alice's chain that was accepted by Bob.
- $e' := Null$: the MAC value received in the current session, supposedly from Alice.
- $M' := Null$: the message received in the current session, supposedly from Alice.
- a one-trit flag, to distinguish the program states **B0**, **B1**, and **B2**.

Alice and Bob start in program states **A0** and **B0**. We write commit-message (M, i_A) to indicate that Alice is committing herself to sending the message M to Bob in session i_A. We let T be the maximum amount of time Alice waits to receive a response from Bob, and vice versa.

A0 is executed as follows:

```
If $i_A \leq 0$ then **Abort**.
Receive input $(M)$ and commit-message$(M, i_A)$.
Compute $e_{i_A} := \mathrm{MAC}_{a_{i_A}}(i_A \| M)$.
Send $[e_{i_A}, M]$ to Bob and **goto A1**.
```

B0 is executed as follows:

```
If $i_B \leq 0$ then **Abort**.
Wait to receive $[e', M']$, then **goto B1**.
```

B1 has the following description:

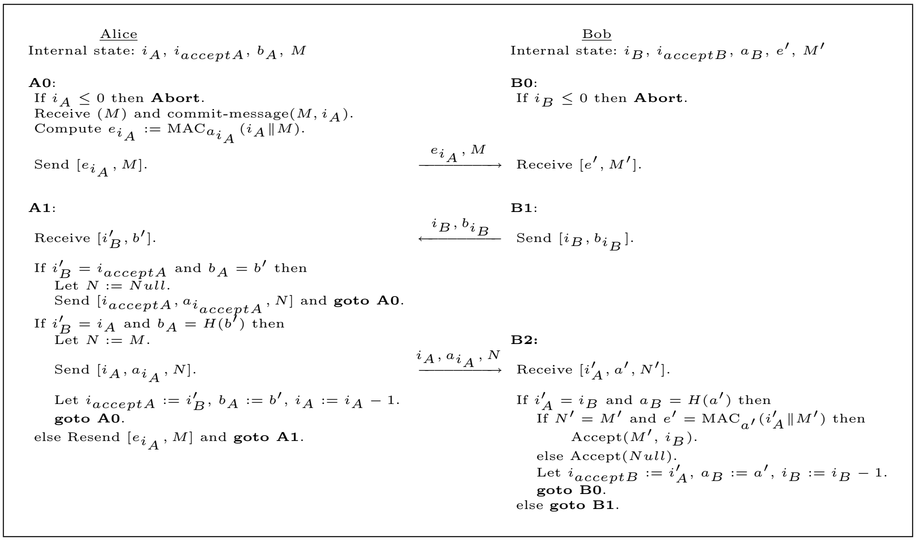
```
Send $[i_B, b_{i_B}]$ to Alice and **goto B2**.
```

A1 is performed in the following manner:

```
Wait at most time $T$ to receive $[i'_B, b']$.
If $[i'_B, b']$ is received, then
    If $i'_B = i_{acceptA}$ and $b_A = b'$ (Bob has not received the last flow of the previous session) then
        Let $N := Null$.
        Send $[i_{acceptA}, a_{i_{acceptA}}, N]$ and **goto A0**.
    If $i'_B = i_A$ and $b_A = H(b')$ then (Alice and Bob seem to be synchronized.)
        Let $N := M$.
        Send $[i_A, a_{i_A}, N]$ to Bob.
        Let $i_{acceptA} := i'_B$, $b_A := b'$ and $i_A := i_A - 1$. (Alice updates her state.)
        **goto A0**.
    else Resend $[e_{i_A}, M]$ to Bob and **goto A1**.
If timeout then
Resend $[e_{i_A}, M]$ to Bob and **goto A1**.
```

B2 is performed as follows:

```
Wait at most time $T$ to receive $[i'_A, a', N']$.
If $[i'_A, a', N']$ is received, then
    If $i'_A = i_B$ and $a_B = H(a')$ then (Alice and Bob seem to be synchronized.)
        If $N' = M'$ and $e' = \mathrm{MAC}_{a'}(i'_A \| M')$ then
            Accept$(M', i_B)$.
        else Accept$(Null)$.
        Let $i_{acceptB} := i'_A$, $a_B := a'$ and $i_B := i_B - 1$. (Bob updates his state.)
        **goto B0**.
    else **goto B1**.
If timeout, then **goto B1**.
```



```
Alice
Internal state: $i_A$, $i_{acceptA}$, $b_A$, $M$

A0:
 If $i_A \le 0$ then Abort.
 Receive $(M)$ and commit-message$(M, i_A)$.
 Compute $e_{i_A} := \mathrm{MAC}_{a_{i_A}}(i_A \| M)$.

 Send $[e_{i_A}, M]$.                          $\xrightarrow{e_{i_A}, M}$

A1:

 Receive $[i'_B, b']$.                         $\xleftarrow{i_B, b_{i_B}}$

 If $i'_B = i_{acceptA}$ and $b_A = b'$ then
    Let $N := Null$.
    Send $[i_{acceptA}, a_{i_{acceptA}}, N]$ and goto A0.
 If $i'_B = i_A$ and $b_A = H(b')$ then
    Let $N := M$.

    Send $[i_A, a_{i_A}, N]$.                  $\xrightarrow{i_A, a_{i_A}, N}$

    Let $i_{acceptA} := i'_B$, $b_A := b'$, $i_A := i_A - 1$.
    goto A0.
 else Resend $[e_{i_A}, M]$ and goto A1.
```

```
Bob
Internal state: $i_B$, $i_{acceptB}$, $a_B$, $e'$, $M'$

B0:
 If $i_B \le 0$ then Abort.

 Receive $[e', M']$.

B1:

 Send $[i_B, b_{i_B}]$.

B2:

 Receive $[i'_A, a', N']$.

 If $i'_A = i_B$ and $a_B = H(a')$ then
   If $N' = M'$ and $e' = \mathrm{MAC}_{a'}(i'_A \| M')$ then
       Accept$(M', i_B)$.
   else Accept$(Null)$.
   Let $i_{acceptB} := i'_A$, $a_B := a'$, $i_B := i_B - 1$.
   goto B0.
 else goto B1.
```

Fig. 3. Our Proposed Message Recognition Protocol (Common Case)

Figure 3 illustrates the main steps of this protocol. For simplicity, the instructions on what to do in case one party does not receive any response from the other party is not included in the figure.

If either Alice or Bob receives a message that they did not expect, they are going to ignore it. For instance, while Alice is in state **A1** and is waiting to receive a message of the form (i_B, b), she is going to ignore messages of the form (M') that request for a new session and correspond to state **A0**. Analogously, when Bob is in state **B2**, he is waiting for a message of type i_A, a, N. He is going to ignore messages of the form e'_{i_A}, M' since they correspond to state **B0**. In general, each party only acts on received messages that have the expected structure in accordance to their current program state.

When Alice is waiting in state **A1** for Bob to respond, she is set to wait for time T. If she receives a message i'_B, b' in time T, then she processed it in state **A1**, and otherwise, she resends e_{i_A}, M to Bob. Similarly, Bob waits to receive a message i'_A, a', N', supposedly from Alice, for time T. If he does not receive such a message, he resends i_B, b to Alice.

Note that Eve can block the last flow of Alice, i_A, a, N. In this case, Alice has decremented her state, while Bob is waiting to receive i_A, a, N, and possibly resending i_B, b_{i_B} to remind Alice to send him i_A, a, N. However, since Alice has moved her state to **A0**, she will ignore Bob's messages. This may appear to be problematic since Bob is waiting for Alice. However, once Alice is ready to authenticate a new message to Bob, she will be in program state **A1** again, and communication will resume.

4 Security of Our New Message Recognition Protocol

In this section, we begin by listing the required security properties of the hash function H and the message authentication code MAC. Then, we consider different types of possible attacks against our protocol. Finally, we conclude with a theorem which ensures the security of our protocol.

4.1 Security Assumptions

In this section, we list the security assumptions required for this protocol. These definitions are notions defined by Lucks et al. [4].

Definition 1. *Let secret $y_0, y_1, \ldots, y_i$ and known y_{i+1} be chosen such that $y_{i+1} = H(y_i)$, $y_i = H(y_{i-1}), \ldots, y_1 = H(y_0)$. A hash function H is referred to as a* **depth-i preimage resistant (i-PR)** *hash function when it is infeasible to find y' such that $y_{i+1} = H(y')$.*

Definition 2. *Let secret $y_0, y_1, \ldots, y_{i-1}$ and known y_i, y_{i+1} be chosen such that $y_{i+1} = H(y_i)$, $y_i = H(y_{i-1}), \ldots, y_1 = H(y_0)$. A hash function H is* **depth-i second preimage resistant (i-SPR)** *when it is infeasible to find y', $y' \neq y_i$, such that $y_{i+1} = H(y')$.*

Definition 3. *Let secret $y_0, y_1, \ldots, y_i$ and known y_{i+1} be chosen such that $y_{i+1} = H(y_i)$, $y_i = H(y_{i-1}), \ldots, y_1 = H(y_0)$. A message authentication code MAC is* **depth-i existentially unforgeable** *if it is infeasible to mount an existential forgery against MAC_{y_i} in an adaptive chosen message attack scenario.*

4.2 Single-Session Attacks

In this section, we consider attacks that are started and completed in a single session. We assume that Eve has stayed passive all along and she becomes active in the current session for the first time. In case of a successful attack, Bob will accept some message M' in the same session, where M' is not $Null$ and not the message sent by Alice in that session. Since Eve has been passive before this session, we will have $i_A = i_B$ at the start of the session; we let $i := i_A = i_B$ for ease of reference. For the same reason, we have $i_{acceptA} = i_{acceptB} = i + 1$. Furthermore, Alice and Bob will have accepted all the intended keys so far. That is, $a_B = a_{i+1}$ and $b_A = b_{i+1}$.

We now want to exhaustively list all possible single-session attacks. We follow the notation of [2] in referring to different orderings of the flows. In each attack, the adversary sends a flow to either Alice or Bob and receives a flow in response. This notation labels a flow by **A** if the recipient is Alice, or by **B** when the recipient is Bob. For instance, the following attack scenario corresponds to the attack type of ABAB:

- **A**: Eve sends M to Alice and she responds with e_{i_A}, M.
- **B**: Eve sends e', M' to Bob and he replies with i_B, b_{i_B}.
- **A**: Eve sends i'_B, b' to Alice and receives i_A, a_{i_A}, N from her.
- **B**: Eve sends i'_A, a', N' to Bob.

The number of distinct attacks against a three flow protocol is proved to be $\binom{4}{2} = 6$ in [2]. These attacks are denoted AABB, ABBA, BABA, ABAB, BBAA, and BAAB. We will look at these different attacks separately. We stress that [2] formally proves this list to be an exhaustive list of all possible types of attacks.

One can show that the BABA attack scenario can be reduced to the ABBA attack. That is, if an adversary Oscar can mount a successful attack of type BABA, then Eve can use Oscar and succeed in the ABBA attack scenario. Similarly, we can show that the BAAB and ABBA attack scenarios are reduced to the ABAB case. We outline these three reductions in the Appendix. It remains to analyze the other three attack scenarios, namely AABB, BBAA, and ABAB. We will reduce a successful adversary in these attacks to a player who can mount a depth-i existential forgery or can find depth-i preimages or depth-i second preimages.

Attack of Type AABB

Figure 4 depicts an attack of type AABB.

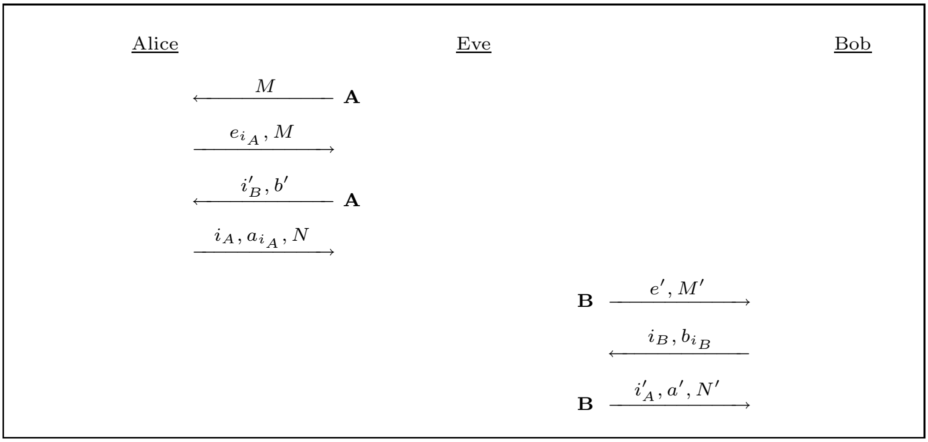


Fig. 4. Attack of Type AABB

If $i'_A \neq i_B$, Bob will not accept any messages. Since $i_A = i_B = i$, Eve has to set $i'_A := i_A$ in order to succeed. Moreover, Alice reveals i_A and a_{i_A} only if b' is verified; that is, if $b_A = H(b')$ (note that $b_A = b_{i+1}$, as discussed before).

Eve first interacts with Alice and has to find b' before seeing $b_{i_B} = b_i$. This implies that she has found a preimage of $b_A = b_{i+1}$. This exactly translates to the notion of i-PR defined in Def. 1.

Attack of Type BBAA

Figure 5 illustrates the attack of type BBAA.

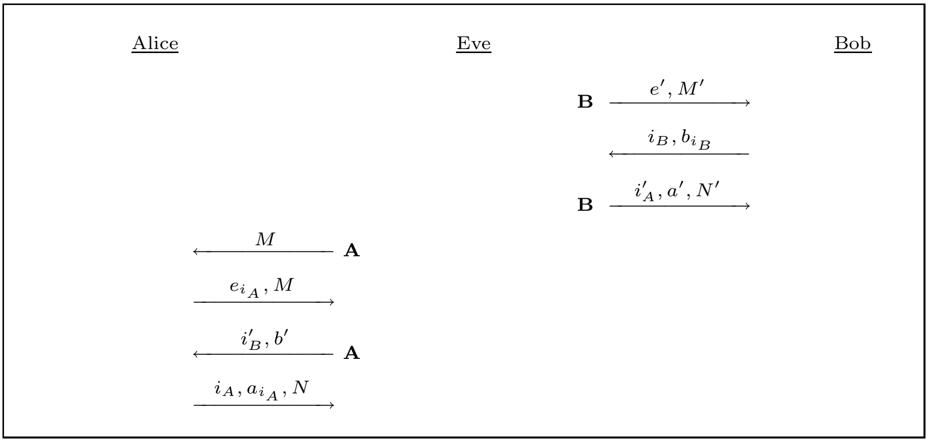


Fig. 5. Attack of Type BBAA

Alice tries to deceive Bob before she starts interacting with Alice. In order to succeed, Eve needs to present Bob with an a' such that $a_B = H(a')$, without having seen $a_{i_A} = a_i$ (note that $a_B = a_{i+1}$, as discussed before). In other words, she is trying to find a preimage of $a_B = a_{i+1}$. If Eve can successfully find such a preimage, the she translates to a successful player who finds depth-i preimages, as defined in Def. 1.

Attack of Type ABAB

Depicted in Fig. 6 is the ABAB attack.

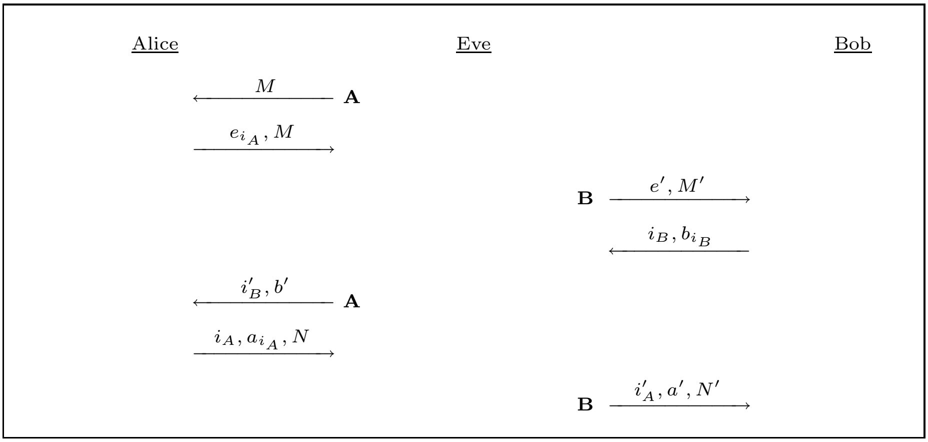


Fig. 6. Attack of Type ABAB

In this scenario, Eve receives $b_{i_B} = b_i$ before she has to send b' to Alice. We analyze the two cases $b' = b_i$ and $b' \neq b_i$ separately.

If $b' \neq b_i$, then it implies that Eve has found a depth-i second preimage of $b_A = b_{i+1}$.

Otherwise, $b' = b_i$. Alice will verify $b' = b_i$ and reveal $a_{i_A} = a_i$. Eve now has two choices. She chooses a' such that either $a' = a_{i_A}$ or $a' \neq a_{i_A}$. If $a' \neq a_{i_A}$, then she has found a depth-i second preimage of $a_{i+1} = a_B$. On other hand, if $a' = a_{i_A}$, then for Eve to succeed, she must set $N' := M'$ and she must have set $e' := \mathrm{MAC}_{a'}(i'_A \| M')$ *before* learning a'. That is, Eve has successfully forged a MAC. This reduces to the notion of depth-i existential forgery defined in Def. 3.

4.3 Multi-session Attacks

Having ruled out the possibility of single-session attacks, we now turn our attention to multi-session attacks. Consider attack scenarios which occur over two or more sessions. In such a case, the adversary becomes active in one session and concludes her attack in one of the following sessions. In case of a successful attack, Bob will accept M' in the last session of the attack, where M' is not $Null$ and not the message sent by Alice in that session.

Just before Eve becomes active, similar to the single-session attack scenario discussed above, we must have $i_A = i_B$ and $i_{acceptA} = i_{acceptB} = i_A + 1$. We again let $i := i_A = i_B$ for ease of reference. Moreover, all of the intended keys will have been accepted to this point, so as a result, $a_B = a_{i+1}$ and $b_A = b_{i+1}$.

We now assume that during session i, Eve becomes active by initiating a flow with either Alice or Bob, or changing the information sent by them. Since we are considering multi-session attacks, the attack should not entirely take place in one session. As a result, Eve is not making Bob accept her message M' immediately after she becomes active. The following three cases can happen once Eve becomes active:

Case 1. Bob is not engaged right away. That is, Eve first interacts with Alice.

Case 2. Bob is engaged right away and he outputs the message M, sent by Alice.

Case 3. Bob is engaged right away and he outputs $Null$.

We discuss each case separately.

Case 1. Let us assume that Eve first interacts with Alice and does not engage Bob. In order for Alice to conclude her session, she must receive i'_B, b' such that $i'_B = i$ and $b_{i+1} = H(b')$. Otherwise, Alice will detect that something is going on, hence, she will not reveal i, a_i and, instead, will resend e_i, M. If Eve wants to remain undetected and be able to continue with her attack, she needs to send i'_B, b' such that $i'_B = i$ and $b_{i+1} = H(b')$. This means that Eve has found a depth-i preimage of b_{i+1}.

Case 2. Now assume that Bob is engaged and he outputs the message M, sent by Alice. That is, on input (M), Alice has sent e_i, M to Bob. Since Bob accepts M at the end, it means that he, indeed, has received M in the first flow. Moreover, for Bob to accept M, he must receive i'_A, a', N' such that $i'_A = i$, $a_{i+1} = H(a')$, and $N' = M$. There are three different cases to consider here.

- Not having received i, a_i, M from Alice, Eve finds i'_A, a', N' such that $i'_A = i$ and $a_{i+1} = H(a')$. That is, she finds a depth-i preimage of a_{i+1}.
- Having received i, a_i, M from Alice, Eve finds i'_A, a', N' such that $i'_A = i$, $a_{i+1} = H(a')$, and $a_i \neq a'$. That is, she finds a depth-i second preimage of a_{i+1}.
- Eve sets $i'_A, a', N' = i, a_i, M$. That is, Eve relays Alice's last flow. Note that Alice reveals her last flow only if she receives i'_B, b' such that $i'_B = i$ and $b_{i+1} = H(b')$. There are again three cases to consider here. Either Eve has found a depth-i preimage of b_{i+1}, she has found a depth-i second preimage of b_{i+1}, or she has relayed i, b faithfully. In the latter case, Eve has faithfully relayed all messages, and this does not constitute an attack by an active adversary. This contradicts our assumption that Eve first becomes active in session i.

Case 3. Bob is engaged right away and he outputs $Null$. This means that he has received and verified i'_A and a'. There are again three cases to consider. Either Eve has found a depth-i preimage of a_{i+1}, or she has found a depth-i second preimage of a_{i+1}, or i'_A and a' are the correct i, a_i as revealed by Alice. In this last case, Alice and Bob have successfully remained synchronized, but were unable to authenticate the messages they intended to authenticate.

The above discussion concludes that in the session immediately after Eve becomes active, she can only stop Alice and Bob from authenticating the intended message, but she cannot bring them out of their synchronized states unless she is able to solve the depth-i PR or depth-i SPR problems defined in Definitions 1 and 2. Moreover, if Alice and Bob are synchronized at the beginning of a session, then they will end the session in a synchronized state, unless Eve is able to find depth-i preimages or depth-i second preimages.

At the beginning of a multi-session attack, Alice and Bob are synchronized. The above discussion implies that they remain synchronized until the very last session of the attack. We can look at this last session of the attack separately and think of it as a single-session attack. As a result, any multi-session attack translates to a single-session attack, which were already ruled out in Section 4.2.

Note that the adversary can only exhaust Alice's and Bob's values of the hash chain one at a time. That is, she can not make them jump more than one step down the hash chain values.

4.4 Self-recoverability

In this section, we show that once Eve stops interfering with their message flows, Alice and Bob will be able to resume successful communication of recognized messages. Because we have already shown that Alice and Bob remain synchronized in their i values throughout an active attack by Eve (under the security assumptions on H and MAC), we need only show that they do not get "trapped" in a program state, as was the case in the Jane Doe protocol, for example.

We consider the possible combinations of program states which Alice and Bob are in when Eve becomes passive. We first consider the case where Alice is in state **A1**.

- If Alice is in **A1** and Bob is in **B0**, then after time T, Alice will resend $[e_{i_A}, M]$ to Bob, which will cause him to leave state **B0**, and the protocol will continue.
- If Alice is in **A1** and Bob is in **B1**, then Bob will send $[i_B, b_{i_B}]$ to Alice and advance to **B2**, which will cause her to send an appropriate message to Bob, and herself return to **A0**. Bob will return to **B0**, though he may Accept($Null$) if Eve forged the M' which caused Bob to enter the **B1** state. This can of course only affect the first Accept after Eve's interference, however.
- If Alice is in **A1** and Bob is in **B2**, then Alice will be resending useless messages to Bob, and staying in **A1**, but after time T, Bob will return to **B1**, and we proceed as above.

If Alice is in **A0**, then no progress will be made until the next time she tries to send a message to Bob. At that point, Alice will enter state **A1**, and the analysis continues as above.

4.5 Main Theorem

The above discussion concludes the discussion of the security and self-recoverability of the proposed message recognition protocol, and forms the proof of the following theorem.

Security and Self-recoverability Theorem. *A successful adversary against the protocol of Section 3 who efficiently deceives Bob into accepting (M',i), where M' is not $Null$ and Alice did not send M' in session i, implies an efficient algorithm that finds depth-i preimages or depth-i second preimages, or creates depth-i existential forgeries. Moreover, the adversary cannot stop Alice and Bob from successfully executing the protocol unless she is actively disrupting the communication for the lifetime of Alice and Bob.*

5 Comments and Conclusion

We briefly reviewed the definitions and the security model of message recognition protocols in the literature. We looked at the Jane Doe message recognition protocol proposed by Lucks et al. [4] in more detail and described its inability to recover in case of a certain adversarial disruption. In order to overcome the recoverability problem of the Jane Doe protocol, we proposed a new message recognition protocol, which is based on the Jane Doe protocol. This new protocol incorporates a resynchronization technique within itself and, hence, provides self-recoverability. Finally, we formally proved the security of our protocol.

It should be noted that our new protocol is somewhat less efficient than the Jane Doe protocol in that each message M is transmitted twice (in the first flow, and again in the third flow of Figure 3). This would not be a problem if the communication channel is inexpensive. However, it (roughly) doubles the power consumption as compared to the Jane Doe protocol if messages are large. If this creates a problem, it would be possible to modify our protocol by sending $N = H(M)$ in the third flow instead of $N = M$. Then Bob checks that $N' = H(M')$ instead of $N' = M'$.

Acknowledgements

We would like to thank Natural Sciences and Engineering Research Council of Canada (NSERC) and Mathematics of Information Technology and Complex Systems (MITACS) for supporting this research.

References

1. Anderson, R., Bergadano, F., Crispo, B., Lee, J.-H., Manifavas, C., Needham, R.: A new family of authentication protocols. In: ACMOSR: ACM Operating Systems Review, vol. 32, pp. 9–20 (1998)
2. Gehrmann, C.: Multiround unconditionally secure authentication. Designs, Codes, and Cryptography 15(1), 67–86 (1998)
3. Hammell, J., Weimerskirch, A., Girao, J., Westhoff, D.: Recognition in a low-power environment. In: ICDCSW 2005: Proceedings of the Second International Workshop on Wireless Ad Hoc Networking (WWAN), Washington, DC, USA, 2005, pp. 933–938. IEEE Computer Society Press, Los Alamitos (2005)
4. Lucks, S., Zenner, E., Weimerskirch, A., Westhoff, D.: Entity recognition for sensor network motes. In: GI Jahrestagung (2), pp. 145–149 (2005)
5. Lucks, S., Zenner, E., Weimerskirch, A., Westhoff, D.: Concrete security for entity recognition: The Jane Doe protocol. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 158–171. Springer, Heidelberg (2008)
6. Mashatan, A., Stinson, D.R.: A New Message Recognition Protocol For Ad Hoc Pervasive Networks. In: Franklin, M.K., Hui, L.C.K., Wong, D.S. (eds.) CANS 2008. LNCS, vol. 5339, pp. 378–394. Springer, Heidelberg (2008)
7. Mashatan, A., Stinson, D.R.: Interactive two-channel message authentication based on Interactive-Collision Resistant hash functions. Int. J. Inf. Secur. 8(1), 49–60 (2009)
8. Mashatan, A., Stinson, D.R.: Recognition in ad hoc pervasive networks. Technical Report 2008-12, Centre for Applied Cryptographic Research (CACR), University of Waterloo, Canada (2008)
9. Mitchell, C.J.: Remote user authentication using public information. In: Paterson, K.G. (ed.) Cryptography and Coding 2003. LNCS, vol. 2898, pp. 360–369. Springer, Heidelberg (2003)
10. Pasini, S., Vaudenay, S.: An optimal non-interactive message authentication protocol. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 280–294. Springer, Heidelberg (2006)

11. Weimerskirch, A., Westhoff, D.: Zero common-knowledge authentication for pervasive networks. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 73–87. Springer, Heidelberg (2004)

A Reducing Three Single-Session Attacks

As was described in Section 4, Gehrmann [2] formally proves that there are only six possible types of single-session attack against the protocol of Figure 3. We analyzed the AABB, BBAA, and ABAB attacks in that section. Here we examine the remaining three attacks: BABA, BAAB, and ABBA. The BABA attack is reduced to the ABBA attack. Then, the ABBA attack is reduced to the ABAB attack. Finally, the BAAB attack is also reduced to the ABAB attack. This concludes the analysis of the six different attack scenarios.

A.1 Reducing the BABA Attack to an ABBA Attack

The ABBA attack scenario, depicted in Fig. 7, is as follows:

- **A**: Oscar sends M to Alice and receives e_{i_A}, M from her.
- **B**: Oscar sends e', M' to Bob and he sends i_B, b_{i_B}.
- **B**: Oscar sends i'_A, a', N' to Bob.
- **A**: Oscar sends i'_B, b' to Alice and she replies with i_A, a_{i_A}, N.

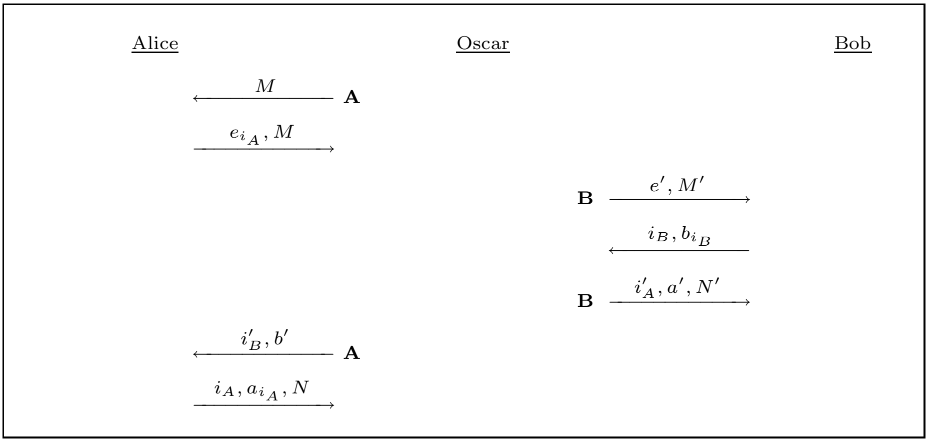


Fig. 7. Attack of Type ABBA

On the other hand, the BABA attack scenario, illustrated in Fig. 8, is as follows:

- **B**: Oscar sends e', M' to Bob and he sends i_B, b_{i_B}.
- **A**: Oscar sends M to Alice and receives e_{i_A}, M from her.

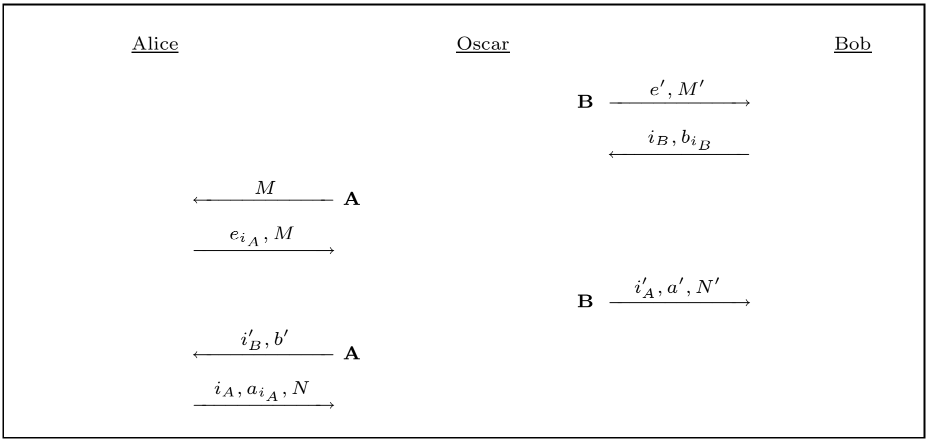


Fig. 8. Attack of Type BABA

- **B**: Oscar sends i'_A, a', N' to Bob.
- **A**: Oscar sends i'_B, b' to Alice and she replies with i_A, a_{i_A}, N.

These two attack scenarios differ in the order of the first two steps and are identical otherwise. In the BABA attack scenario, Oscar commits to e' and M' before receiving e_{i_A}. Note that knowing e_{i_A} could possibly help him in choosing e'. On the other hand, Oscar receives i_B and b_{i_B} before sending M. The adversary knows the value of i_B. Moreover, the choice of M is independent of the value of b_{i_B}. In other words, knowing b_{i_B} is not going to help the adversary in choosing M. Hence, if Oscar can win in the BABA attack scenario by first committing to e' and M' and then receiving e_{i_A}, then he can win the ABBA attack scenario with the same values M, M', and e.

A.2 Reducing the ABBA Attack to an ABAB Attack

Recall the ABAB attack scenario from Section 4:

- **A**: Oscar sends M to Alice and receives e_{i_A}, M from her.
- **B**: Oscar sends e', M' to Bob and he sends i_B, b_{i_B}.
- **A**: Oscar sends i'_B, b' to Alice and she replies with i_A, a_{i_A}, N.
- **B**: Oscar sends i'_A, a', N' to Bob.

The ABBA attack differs from the ABAB attack in the order of the last two steps. In the ABAB attack, Oscar receives i_A, a_{i_A}, N from Alice, and then he has to send i'_A, a', N' to Bob. Knowing i_A, a_{i_A}, N can help him choose a winning i'_A, a', N', whereas in the ABBA attack scenario, Oscar sends i'_A, a', N' before seeing i_A, a_{i_A}, N. If Oscar has a winning strategy in the ABBA attack scenario, then using the same values of i'_A, a', N', he will win the ABAB attack scenario.

A.3 Reducing the BAAB Attack to an ABAB Attack

The BAAB attack scenario is as follows:

- **B**: Oscar sends e', M' to Bob and he sends i_B, b_{i_B}.
- **A**: Oscar sends M to Alice and receives e_{i_A}, M from her.
- **A**: Oscar sends i'_B, b' to Alice and she replies with i_A, a_{i_A}, N.
- **B**: Oscar sends i'_A, a', N' to Bob.

Figure 9 depicts this attack.

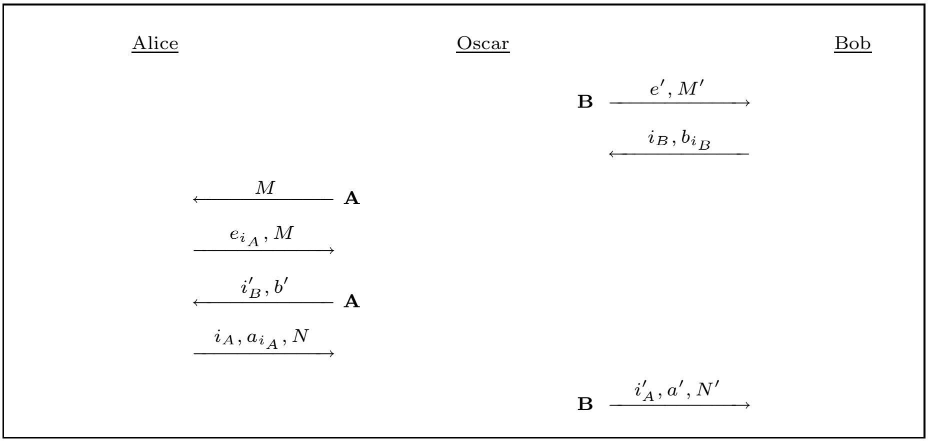


Fig. 9. Attack of Type BAAB

The analysis of this case is analogous to that of Section A.1. The BAAB attack scenario differs from the ABAB attack scenario in the order of the first two steps. In the BAAB attack scenario, Oscar has to commit to e' and M' before seeing e_{i_A}. Although Oscar receives i_B and b_{i_B} before sending M, these values are independent of the choice of M. That is, seeing b_{i_B} is not going to help the adversary in choosing M. Hence, a winning strategy in the BAAB attack scenario reduces to a winning strategy in the ABAB attack scenario.

Breaking Two k-Resilient Traitor Tracing Schemes with Sublinear Ciphertext Size[*]

MoonShik Lee, Daegun Ma, and MinJae Seo

Department of Mathematical Sciences and ISaC-RIM,
Seoul National University, Seoul, 151-747, Korea
{kafa04,madgun7,morion81}@snu.ac.kr

Abstract. In 2004, Matsushita and Imai proposed a k-resilient public-key traitor tracing scheme which has sublinear ciphertext size $4k + 2 + (n/2k)$ with efficient black-box tracing against self-defensive pirates, where n, k are the total number of subscribers and the maximum number of colluders. After that, in 2006, they presented a hierarchical key assignment method to reduce the ciphertext size into $4k + 5 + \log(n/2k)$ by combining a complete binary tree with the former scheme.

In this paper, we show that the proposed schemes are vulnerable to our attack which makes pirate keys able to avoid the black-box tracing. Their schemes are based on multiple polynomials and our attack use a combination between different polynomials. The latter scheme can be broken by other attacks which use secret values of the key generation polynomial or use partial keys.

Keywords: cryptanalysis, public-key traitor tracing, black-box tracing, self-defensive pirates, linear attack.

1 Introduction

In modern times, enormous digital contents are transmitted over the various media through encryption and they can be decrypted only by the legitimate subscribers. But some of the subscribers may collude to make pirate decoders and distribute them. In the traitor tracing schemes, at least one of the traitors can be traced and this traceability is required in various contents delivery systems such as satellite broadcast, DMB, pay-TV, DVD, online database and so on.

The first traitor tracing scheme was introduced by Chor et al. [3] in 1994, which was inefficient and only the system manager could encrypt a message. In 1998, Kurosawa and Desmedt [5] proposed a polynomial based *public-key* tracing scheme, where any data supplier could encrypt a message. In 1999, Boneh and Franklin [1] proposed another public-key traitor tracing scheme with *black-box tracing* algorithm, where the tracer could reveal at least one of the traitors without opening the pirate decoder by using it as a black box.

[*] This research are partially supported by BK21 project and Korea Research Council of Fundamental Science & Technology.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 238–252, 2009.

If a pirate decoder is able to detect a tracing, then it may take some reactions such as erasing the key. In 2001, Kiayias and Yung [4] considered crafty pirates and categorized them into four types, from type-0 to type-3, according to their capabilities(resettable vs. history recording, available vs. abrupt). They also proposed a generic tracing technique of hybrid colorings and schemes of type-2 and type-3 tracing.

In 2004, Matsushita and Imai [6] proposed a k-resilient public-key traitor tracing scheme with efficient black-box tracing against the type-2(resettable and abrupt) pirate decoder. In this scheme, subscribers are split into $\ell(= n/2k)$ sets and a broadcaster transmits a header H with $4k + \ell + 2$ elements, from which each subscriber in the i-th subscriber set extracts a corresponding header H_i for decrypting a session key. Although it has a limit of the collusion size k, they argued that the scheme achieved the efficient black-box tracing with sublinear ciphertext size.

In 2006, the same authors suggested another new idea of combining a tree structure with the scheme of [6] to reduce the ciphertext size [7]. They positioned each subscriber set in the leaf node of a complete binary tree and assigned several keys to a subscriber. They considered three methods of constructing the key generation polynomial, and by the last method which is called a hierarchical key assignment method, they could reduce the ciphertext size from $4k + \ell + 2$ to $4k + 5 + \log \ell$. Comparing with the fully resilient scheme [2] with $6\sqrt{n}$ header size, this scheme is more efficient when k is not so large.

Our contribution. In this paper, we break the two k-resilient schemes of [6] and [7], which have been the most efficient public-key black-box traitor-tracing schemes. An important characteristic of their schemes is that they use multiple polynomials which are interrelated. We newly introduce a variant of linear attack of combining two polynomials to construct pirate keys which cannot be traced. Furthermore, the latter scheme has another problem in adapting a tree structure, which leads to untraceable piracy through the computation of secret values or use of partial keys.

Organization. This paper is organized as follows. In Section 2, we briefly describe the schemes in [6] and [7]. In Section 3, we show our attack on the scheme of [6]. In section 4, we point out that the three methods of [7] are broken by our attack. Finally, in section 5, we conclude our paper.

2 Preliminaries

In this Section, we review the categorization of pirate decoders [4] and the schemes of [6] and [7], which are the targets of our analyses.

2.1 Models of Pirate Decoders

In 2001, Kiayias and Yung [4] categorized pirate decoders by the following two criterion.

Resettable vs. history recording : Resettable decoders can be reset to their initial state by the tracer after each test. History recording pirate decoders remember the previous inputs and may use that information to detect the tracing.

Available vs. abrupt(self-defensive) : Abrupt or self-defensive pirate decoders can take some counter-actions against the tracing if the decoder detects it. Available pirate decoder is a device that does not take such reactions.

From this, they suggested the following 4 types of pirate decoders.

type 0: Available and resettable.
type 1: Available and history recording.
type 2: Abrupt and resettable.
type 3: Abrupt and history recording.

In the schemes of [6] and [7], they considered black-box traceability against type-2 decoders, which would be simply expressed by *type-2 tracing*. Although they didn't state it explicitly, they also assumed that a tracer knew the reaction mechanism, saying that the reaction was triggered once when a tracing was detected.

2.2 Common Parameters of the Schemes

Let n, k denote the total number of subscribers and the maximum number of colluders in a coalition, respectively. Let p, q be the primes s.t. $q \mid p-1$ and $q \geq n+2k$. Let g be a qth root of unity over $\mathbb{Z}_p^*$ and G_q be the subgroup of $\mathbb{Z}_p^*$ of order q. Let $\mathcal{U}$ be a set of subscribers ($\mathcal{U} \subseteq \mathbb{Z}_q^*$). All of the participants agree on p, q and g. Split $\mathcal{U}$ into ℓ disjoint subsets $\mathcal{U}_0, \dots, \mathcal{U}_{\ell-1}$. For notational simplicity, we assume that $|\mathcal{U}_i| = 2k$, $\mathcal{U}_i = \{u_t | 2ki+1 \leq t \leq 2k(i+1)\}$ for $0 \leq i \leq \ell-1$ and $n = 2k\ell$.

2.3 The Scheme of [6]

In 2004, Matsushita and Imai proposed an efficient type-2 black-box tracing scheme with sublinear ciphertext size[6]. We briefly summarize this scheme in different form with minor corrections. Let a *valid input* denote a header for the normal broadcast and an *invalid input* denote a header for the black-box tracing.

Key generation. Choose $a_0, \dots, a_{2k-1}, c_0, \dots, c_{\ell-1} \in_R \mathbb{Z}_q$ and compute the public key e by

$$e = (g, g^{a_0}, \dots, g^{a_{2k-1}}, g^{c_0}, \dots, g^{c_{\ell-1}}).$$

The private key for a subscriber $u \in \mathcal{U}_i$ is generated as $(u, i, f_i(u))$ where

$$f_i(u) = \sum_{j=0}^{2k-1} a_{i,j} u^j \bmod q, \quad a_{i,j} = \begin{cases} a_j & (j \neq i \bmod 2k), \\ c_i & (j = i \bmod 2k). \end{cases} \tag{1}$$

Encryption. Select a session key $s \in_R G_q$ and random numbers $R_0, R_1 \in_R \mathbb{Z}_q$. Build a header $H = (H_0, \ldots, H_{\ell-1})$ by repeating the following procedure for $0 \le i \le \ell - 1$. Set $r_i \in_R \{R_0, R_1\}$, and compute $H_i = (\hat{h}_i, h_{i,0}, \ldots, h_{i,2k-1})$ where

$$\hat{h}_i = g^{r_i}, \quad h_{i,j} = \begin{cases} g^{a_j r_i} & (j \ne i \bmod 2k), \\ s g^{c_i r_i} & (j = i \bmod 2k). \end{cases} \tag{2}$$

Decryption. Suppose that $u \in \mathcal{U}_i$. The subscriber u can compute the session key s from H_i as follows,

$$\left(\frac{\prod_{j=0}^{2k-1} h_{i,j}{}^{u^j}}{\hat{h}_i{}^{f_i(u)}} \right)^{\frac{1}{u^i}} = \left(s^{u^i} \frac{\prod_{j=0}^{2k-1} g^{a_{i,j} u^j r_i}}{g^{f_i(u) r_i}} \right)^{\frac{1}{u^i}} = s.$$

Black-box tracing. For $1 \le t \le n$, repeat the following procedure. Set $\mathcal{X} := \{u_1, \ldots, u_t\}$ as a set of revoked subscribers and $ctr_t = 0$. Find integers t_1, t_2 s.t. $t = 2kt_1 + t_2$ where $0 \le t_1 \le \ell - 1$ and $1 \le t_2 \le 2k$. Repeat the following test m times. In each test, the session key s and R_0, R_1 are chosen randomly.

1. Build the header $H = (H_0, \ldots, H_{\ell-1})$ through the following procedure.

   A: If $t_2 = 2k$, then choose $r_i \in_R \{R_0, R_1\}$ for $0 \le i \le \ell - 1$.

   A-a: For each $0 \le i \le t_1$, select a random number $z_i \in_R \mathbb{Z}_q$, compute $H_i = (\hat{h}_i, h_{i,0}, \ldots, h_{i,2k-1})$ where

   $$\hat{h}_i = g^{r_i}, \quad h_{i,j} = \begin{cases} g^{a_j r_i} & (j \ne i \bmod 2k), \\ g^{z_i} & (j = i \bmod 2k). \end{cases} \tag{3}$$

   A-b: For each $t_1 < i \le \ell - 1$, compute H_i in the same way as (2).

   B: If $t_2 \ne 2k$, then choose $r_i \in_R \{R_0, R_1\}$ for $0 \le i < t_1$ and set $r_{t_1} = R_1$, $r_i = R_0$ for $t_1 < i \le \ell - 1$.

   B-a: For each $0 \le i < t_1$, compute H_i in the same way as (3).

   B-b: For each $t_1 < i \le \ell - 1$, compute H_i with $r_i = R_0$ in the same way as (2).

   B-c: For $i = t_1$, let $x_1 := u_{t+1}, \cdots, x_{2k-t_2} := u_{2k(t_1+1)}$ and choose distinct random numbers $x_j \in \mathbb{Z}_q^* \setminus \mathcal{U}$ for $2k - t_2 < j \le 2k - 1$. Find $d_0, \ldots, d_{2k-1}$ s.t. $\sum_{j=0}^{2k-1} d_j x_\alpha^j = 0$ for all $1 \le \alpha \le 2k - 1$ and compute $H_i = (\hat{h}_i, h_{i,0}, \ldots, h_{i,2k-1})$ where

   $$\hat{h}_i = g^{R_1}, \quad h_{i,j} = \begin{cases} g^{d_j + a_j R_1} & (j \ne i \bmod 2k), \\ s g^{d_i + c_i R_1} & (j = i \bmod 2k). \end{cases} \tag{4}$$

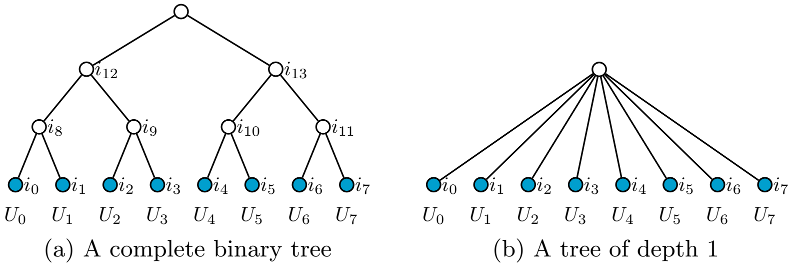
   We call this step of **B-c** the *fine revocation.*

2. Give H to the pirate decoder and observe its output.
3. If it decrypts correctly, then increment ctr_t by one. If a self-defensive reaction is triggered, then decide that the subscriber u_t is a traitor.

Finally, find an integer t s.t. $ctr_{t-1} - ctr_t$ is the maximum and then decide that the subscriber u_t is a traitor, where $ctr_0 = m$.

2.4 The Schemes of [7]

They applied the complete subtree method [8] to reduce the ciphertext size of [6]. By considering a tree T with ℓ leaves they interpreted the scheme of [6] as a depth-1 case (Fig. 1(b)) and generalized it (Fig. 1(a)).


i_{12}
i_{13}
i_8
i_9
i_{10}
i_{11}
i_0 i_1 i_2 i_3 i_4 i_5 i_6 i_7
U_0 U_1 U_2 U_3 U_4 U_5 U_6 U_7
i_0 i_1 i_2 i_3 i_4 i_5 i_6 i_7
U_0 U_1 U_2 U_3 U_4 U_5 U_6 U_7

(a) A complete binary tree (b) A tree of depth 1

Fig. 1. Structure of T ($\ell = 8$)

Let T be a complete binary tree with ℓ leaves, $\mathcal{N}_T$ be a set of the all nodes except the root and $\mathcal{L}_T(\subset \mathcal{N}_T)$ be a set of leaf nodes. For a leaf node $v \in \mathcal{L}_T$, $\mathcal{U}_v$ is a corresponding subscriber set and for a non-leaf node $v \in \mathcal{N}_T \setminus \mathcal{L}_T$, we define $\mathcal{U}_v = \bigcup \mathcal{U}_w$ where w's are the leaves of the subtree rooted at v. For a subscriber u, let $V(u)$ be a set of nodes corresponding to subscriber sets including u i.e., $V(u) = \{v \in \mathcal{N}_T | u \in \mathcal{U}_v\}$. This definition can be naturally extended to the list of subscribers such a way of $V(u, v) = V(u) \cup V(v)$. For a node v, we represent the depth by $\delta(v)$ or just δ.

For notational simplicity, we number the nodes from 0 to $2\ell - 3$ and identify αth node with the index itself, i.e. we use the notation $\mathcal{U}_\alpha := \mathcal{U}_{i_\alpha}$ where $i_\alpha \in \mathcal{N}_T$. For example, in Fig. 1(a), T has $\ell = 8$ leaves and $\mathcal{N}_T = \{0, \ldots, 13\}$, $\mathcal{L}_T = \{0, \ldots, 7\}$, $\mathcal{U}_8 = \mathcal{U}_0 \cup \mathcal{U}_1$, for $u_1 \in \mathcal{U}_0$ and $u_{8k} \in \mathcal{U}_3$, $V(u_1) = \{0, 8, 12\}$, $V(u_{8k}) = \{3, 9, 12\}$ and $V(u_1, u_{8k}) = \{0, 3, 8, 9, 12\}$.

They considered two simple extension methods and proposed one resulting method of constructing key generation polynomials. To indicate each method we use the following notation.

Method 1. The first method of the simple extension,
Method 2. The second method of the simple extension,
Method 3. The third method of their resulting suggestion.

In the following, we summarize their methods. The differences between them are originated from the way of defining the key generation polynomials. Since they stated the full scheme only for the resulting method, so do we.

Key generation

Method 1. Polynomials are generated from a single system, i.e. for $0 \leq v \leq 2\ell - 3$, $f_v(x)$ is defined by the equation (1) in the same way as [6] as well as the public key e. The private keys of a subscriber u are represented as $K_u = \{(u, v, f_v(u)) | v \in V(u)\} = \{(u, v, f_v(u)) \mid v \in \mathcal{N}_T \text{ s.t. } u \in \mathcal{U}_v\}$. For example, in Fig. 1(a), if $u \in \mathcal{U}_0$ then $K_u = \{(u, 0, f_0(u)), (u, 8, f_8(u)), (u, 12, f_{12}(u))\}$.

Method 2. Polynomials are generated from plural systems according to each level, i.e., a $f_v^{(\delta)}(x)$ is generated from the δth system where v is a node at depth δ. Let $e^{(\delta)}$ be a public key corresponding to $f_v^{(\delta)}(x)$. The private keys of a subscriber u are represented as $K_u = \{(u, v, f_v^{(\delta)}(u)) \mid v \in V(u)\}$.

Method 3. Choose $a_i, b_i, c_v, \lambda_v \in_R \mathbb{Z}_q$ for $0 \le i \le 2k-1$ and $0 \le v \le 2\ell-3$. Compute a public key e as follows,

$$e = (g, g^{a_0}, \ldots, g^{a_{2k-1}}, g^{c_0}, \ldots, g^{c_{2\ell-3}}, g^{\lambda_0}, \ldots, g^{\lambda_{2\ell-3}}).$$

Define key generation polynomials, $A_v(x), B(x)$ as follows,

$$A_v(x) = \sum_{i=0}^{2k-1} (a_{v,i} - \lambda_v b_i) x^i \bmod q, \quad B(x) = \sum_{i=0}^{2k-1} b_i x^i \bmod q, \tag{5}$$

where $a_{v,i}$ is the same with the equation (1). The private key K_u of a subscriber u is represented as $K_u = \{(u, v, A_v(u), B(u)) \mid v \in V(u)\}$.

Encryption. Using the following Sel procedure, a broadcaster selects $\log \ell + 1$ nodes and executes Encryption in the previous Subsection for the selected nodes. For $H = (H_{v_1}, \ldots, H_{v_{\log \ell+1}})$, we denote the selected nodes by $V(H)$. For Method 2 and Method 3, it is required additional relations.

Sel. Select $\log \ell + 1$ nodes $v_1, \ldots, v_{\log \ell+1}$ including two leaves which satisfy the condition that $\bigsqcup_{i=1}^{\log \ell+1} \mathcal{U}_{v_i} = \mathcal{U}$, where $\bigsqcup$ means the disjoint union. Note that only one node is selected for each level except the leaf nodes and $|V(u) \cup V(H)| = 1$ for any subscriber u and any header H.

Method 2. $e^{(\delta)}$ is used as a public key when computing H_v where v is a node at depth δ.

Method 3. For each v, $\bar{h}_v = g^{\lambda_v r_v}$ is additionally included in H_v.

Decryption. This is described for only Method 3.
For a subscriber u and a header H, suppose that $V(u) \cap V(H) = v$. Then u computes the session key s by using $(u, v, A_v(u), B(u)) \in K_u$ from H_v as follows,

$$\left(\frac{\prod_{j=0}^{2k-1} h_{v,j}^{\;u^j}}{\hat{h}_v^{A_v(u)} \bar{h}_v^{B(u)}} \right)^{\frac{1}{u^v}} = \left(s^{u^v} \frac{\prod_{j=0}^{2k-1} g^{a_{v,j} u^j r_v}}{g^{(A_v(u) + \lambda_v B(u)) r_v}} \right)^{\frac{1}{u^v}} = s.$$

Black-box tracing. The procedure is similar to the Black-box tracing in the previous Subsection except that $\log \ell + 1$ nodes are chosen in Sel procedure and the fine revocation is executed only on a leaf node $v \in V(H) \cap \mathcal{L}_T$. We omit the detailed description.

3 A Flaw on the Scheme of [6]

In [6], authors asserted that the black-box tracing algorithm would work for any type-2 pirate decoder. In this Section we will show that this is not true by our attack which is a variant of the linear attack. In particular, in the Theorem 2 they followed the logic:

> If the subscriber u_t is not a traitor, then a pirate decoder cannot distinguish an invalid input of $\mathcal{X} = \{u_1, \ldots, u_{t-1}\}$ from an invalid input of $\mathcal{X} = \{u_1, \ldots, u_t\}$ and therefore $ctr_{t-1} - ctr_t \ll m/n$, where m is the number of tests for each $\mathcal{X}$.

However, this logic has a flaw. This means that although u_t is not a traitor, $ctr_{t-1} - ctr_t > m/n$ may happen so that the black-box tracing outputs the innocent subscriber u_t as a traitor.

In the following Subsection, we show our attack as a variant of linear attack on the proposed scheme. The linear attack was considered in [1,9] for k-resilient schemes based on a polynomial of degree k. Suppose that colluders $x_1, \ldots, x_k$ have private keys $(x_1, f(x_1)), \ldots, (x_k, f(x_k))$ for a polynomial $f(x)$ of degree k. Then a linearly combined vector

$$(\delta_0, \ldots, \delta_k, \varDelta) := (\sum_{j=1}^{t} \mu_j, \sum_{j=1}^{t} \mu_j x_j, \ldots, \sum_{j=1}^{t} \mu_j x_j^k, \sum_{j=1}^{t} \mu_j f(x_j))$$

can be used as a key which is not traced, where $\mu_1, \ldots, \mu_t \in \mathbb{Z}_q$. To resist against the linear attack, the schemes based on single polynomial have raised the degree $\geq 2k-1$. However in the scheme of [6], there are multiple polynomials to be used so that the circumstance is somewhat different. As this peculiar structure affects the black-box tracing, a combination of the keys from different polynomials can be used as a key which cannot be traced.

3.1 A Variant of Linear Attack

Suppose that 2 colluders $x_1 \in \mathcal{U}_i, x_2 \in \mathcal{U}_j$ $(i < j)$ collude to make a *pirate key* K_p. Given two private keys $(x_1, f_i(x_1)), (x_2, f_j(x_2))$, they compute a pirate key

$$K_p = \{K_{i,j}\} := \{(x_1, x_2, f_i(x_1) + f_j(x_2))\}.$$

Note that

$$f_i(x_1) + f_j(x_2) = \sum_{\substack{t=0 \\ t \neq i,j}}^{2k-1} a_t(x_1^t + x_2^t) + (c_i x_1^i + a_i x_2^i) + (a_j x_1^j + c_j x_2^j).$$

In fact, each subscriber's key $(u, f_i(u))$ can be used as a vector $(u, u^2, \ldots, u^{2k-1}, f_i(u))$ in decryption phase, therefore we can also regard the pirate key $K_{i,j}$ as a vector

$$K_{i,j} = (x_1 + x_2, x_1^2 + x_2^2, \ldots, x_1^{2k-1} + x_2^{2k-1}, x_1^i, x_2^j, f_i(x_1) + f_j(x_2)).$$

Proposition 1. *For a given valid input, the pirate decoder with K_p of the above form can compute a session key with probability $\frac{1}{2}$.*

Proof. For a given valid input of $H = (H_0, \ldots, H_{\ell-1})$, from the H_i, H_j, the pirate decoder first computes[1]

$$\begin{aligned}\gamma &:= h_{i,i}^{x_1^i}\ h_{j,i}^{x_2^i}\ h_{i,j}^{x_1^j}\ h_{j,j}^{x_2^j}\\ &= (sg^{c_i r_i})^{x_1^i}(g^{a_i r_j})^{x_2^i}(g^{a_j r_i})^{x_1^j}(sg^{c_j r_j})^{x_2^j}\\ &= s^{x_1^i + x_2^j}\cdot g^{c_i x_1^i r_i + a_i x_2^i r_j + a_j x_1^j r_i + c_j x_2^j r_j}.\end{aligned} \tag{6}$$

If $r_i = r_j$, then it can compute the session key by

$$\begin{aligned}&\left(\gamma \prod_{\substack{t=0\\ t\neq i,j}}^{2k-1} h_{i,t}^{x_1^t + x_2^t} \Big/ \hat{h}_i^{f_i(x_1)+f_j(x_2)}\right)^{\frac{1}{x_1^i + x_2^j}}\\ =&\left(\gamma \prod_{\substack{t=0\\ t\neq i,j}}^{2k-1} (g^{a_t r_i})^{x_1^t + x_2^t} \Big/ g^{(f_i(x_1)+f_j(x_2))r_i}\right)^{\frac{1}{x_1^i + x_2^j}} = s.\end{aligned} \tag{7}$$

Since each r_i is chosen at random from $\{R_0, R_1\}$ for valid inputs, the probability that $r_i = r_j$ is $\frac{1}{2}$ for any $i \neq j$. □

However, the pirate decoder with the decryption probability $\frac{1}{2}$ may not be used in many applications. For these cases we can also consider a pirate decoder with the probability 1 which can be made by 3 colluders. Suppose that 3 colluders $x_1 \in \mathcal{U}_i, x_2 \in \mathcal{U}_j, x_3 \in \mathcal{U}_k (i < j < k)$ with a pirate key $K_p = \{K_{i,j}, K_{i,k}, K_{j,k}\}$. Since a broadcaster selects r_i, r_j, r_k randomly from $\{R_0, R_1\}$ to build a valid input, at least two values of them coincide, therefore the pirate decoder with this key can compute a session key with probability 1 according to the Proposition 1.

3.2 Probability of Untraceability

We now consider the probability that the pirate decoder with a pirate key $K_{i,j}$ $(i < j)$ can decrypt the invalid inputs for the given revocation set $\mathcal{X}$. Note that in the black-box tracing algorithm, the tracer uses this probability to decide whether a traitor is included in the $\mathcal{X}$ or not. Since the black-box algorithm requires the gradual increment of the revoked subscriber set $\mathcal{X}$, we assume that $\mathcal{X} = \{u_1, \ldots, u_t\}$[2], where $t = 2kt_1 + t_2$ for $0 \le t_1 \le \ell - 1$ and $1 \le t_2 \le 2k$.

Let Pr_t denote the probability that the pirate decoder computes a session key correctly from the invalid inputs H for $\mathcal{X} = \{u_1, \ldots, u_t\}$. We consider a pirate key $K_{i,j}$ made by two colluders $x_1 \in \mathcal{U}_i$, $x_2 \in \mathcal{U}_j$ $(i < j)$ at first. Note that the

[1] For easy reading, we drop the 'mod $2k$' in the second subscript of h.

[2] In the black-box tracing phase, a tracer can determine the order of subscriber set to be revoked as well as the order of subscriber to be revoked in a given subscriber set.

[**A-b**] and the [**B-b**] steps are the same as the valid inputs. Then, according to the case in the tracing algorithm, the probability Pr_t for each $1 \leq t \leq n$ becomes:

1. If $\mathcal{X} \cap \mathcal{U}_i = \varnothing$ and $t_2 \neq 2k$, then $Pr_t = 1$, since H_i and H_j are computed according to [**B-b**] and $r_i = r_j = R_0$.

2. If $\mathcal{X} \cap \mathcal{U}_i = \varnothing$ and $t_2 = 2k$, then $Pr_t = \frac{1}{2}$, since H_i and H_j are computed according to [**A-b**] and r_i, r_j are chosen from $\{R_0, R_1\}$ randomly.

3. If $\varnothing \neq \mathcal{X} \cap \mathcal{U}_i \neq \mathcal{U}_i$ and $\mathcal{X} \cap \mathcal{U}_j = \varnothing$ then $Pr_t = 0$, since $H_i = (g^{R_1}, g^{d_0+a_0R_1}, \ldots, sg^{d_i+c_iR_1}, \ldots, g^{d_{2k-1}+a_{2k-1}R_1})$ is computed according to [**B-c**] and H_j is computed according to [**B-b**] with $r_j = R_0$, it becomes $r_i \neq r_j$.

4. If $\mathcal{X} \supset \mathcal{U}_i$ then $Pr_t = 0$, since H_i is computed according to [**A-a**] or [**B-a**] and $h_{i,i} = g^{z_i}$, so the pirate decoder cannot decrypt the session key.

Observe that the Pr_t does not decrease so that their black-box algorithm does not work. However, since the probability vanishes from $t = 2ki+1$, a tracer can know that the first traitor x_1 is positioned in $\mathcal{U}_i$, but cannot know who the traitor is exactly.

Theorem 1. *For the pirate decoder with a pirate key $K_{i,j}$ by two colluders $x_1 \in \mathcal{U}_i$ and $x_2 \in \mathcal{U}_j$ $(i < j)$, they cannot be traced with the probability $1 - \frac{1}{2k}$.*

Proof. It is clear from the above argument. Since there are $2k$ subscribers in $\mathcal{U}_i$, the probability that x_1 can be pointed out as a traitor is $\frac{1}{2k}$. □

We now consider a pirate decoder from 3 colluders, $x_1 \in \mathcal{U}_i, x_2 \in \mathcal{U}_j, x_3 \in \mathcal{U}_k$ where $i < j < k$ with 3 combined keys $K_p = \{K_{i,j}, K_{i,k}, K_{j,k}\}$. Then since $r_i, r_j, r_k \in_R \{R_0, R_1\}$, there must be at least one pair that the random numbers coincide. So, this pirate decoder can always decrypt the valid inputs correctly. The probability that this pirate decoder can decrypt the invalid inputs correctly will be:

1. If $\mathcal{X} \cap \mathcal{U}_i = \varnothing$, then $Pr_t = 1$, since
   (a) If $t_2 \neq 2k$ then H_i, H_j, H_k are computed according to [**B-b**] and $r_i = r_j = r_k = R_0$, so it decrypts all such H correctly.
   (b) If $t_2 = 2k$ then according to [**A-b**] there is one pair that the random numbers coincide, so it decrypts all such H correctly.

2. If $\mathcal{X} \cap \mathcal{U}_i \neq \varnothing$ and $\mathcal{X} \cap \mathcal{U}_j = \varnothing$, then
   (a) If $t_2 \neq 2k$ then $Pr_t = 1$, since H_j, H_k are computed according to [**B-b**] and $r_j = r_k = R_0$.

(b) If $t_2 = 2k$ then $Pr_t = \frac{1}{2}$, since $h_{i,i} = g^{z_i}$ according to [**A-a**], and H_j, H_k are computed according to [**A-b**] and the probability that $r_j = r_k$ is $\frac{1}{2}$.

3. If $\varnothing \neq \mathcal{X} \cap \mathcal{U}_j \neq \mathcal{U}_j$ and $\mathcal{X} \cap \mathcal{U}_k = \varnothing$, then $Pr_t = 0$, since according to [**B-a**], [**B-b**] and [**B-c**], $h_{i,i} = g^{z_i}$ in H_i and H_j, H_k are computed by $r_j = R_1$ and $r_k = R_0$ respectively.

4. If $\mathcal{X} \supset \mathcal{U}_j$, then $Pr_t = 0$, since according to [**B-a**], $h_{i,i} = g^{z_i}$ in H_i and $h_{j,j} = g^{z_j}$ in H_j.

Observe that a tracer can know in which subscriber set $\mathcal{U}_j$ the second traitor x_2 is positioned, but cannot know who the traitor is exactly. From this we can obtain the similar result that this pirate decoder from 3 colluders cannot be traced with the probability $1 - \frac{1}{2k}$. We skip the explanations on the cases that more than 3 colluders attend the piracy.

4 Flaws on the Scheme of [7]

In [7], to decrease the ciphertext size, the authors considered a complete binary tree and bound each subscriber set $\mathcal{U}_i$ to a leaf node for $0 \leq i \leq \ell - 1$. Then they considered three methods of defining key generation polynomials for each node. They argued that the first simple extension method was insecure against collusion attack through solving linear equations[3], the second simple extension method was just inefficient from the viewpoint of the header size and the last hierarchical key assignment method was efficient and secure. However, all of their arguments were flawed.

In this Section, we will show what flaws they have; (1) they are also vulnerable to our attack as shown in Section 3, (2) there is an easy way of extracting secret values which can be used for decryption and (3) there is a structural flaw of easy construction of a non-traceable pirate key.

4.1 A Variant of Linear Attack on the Hierarchical Key Assignment Methods

The first and third methods are also vulnerable to our attack as shown in Section 3. Now consider a case of 2 colluders x_1 and x_2. Suppose that x_1 and x_2 are positioned at the left child and right child node of the root, respectively, or equivalently $V(x_1) \cap V(x_2) = \varnothing$. For the case of $V(x_1) \cap V(x_2) \neq \varnothing$, we will introduce different types of attacks in the following Subsections. To avoid unnecessary repetitions, we describe our attack just on the third method.

[3] But this argument is incorrect. They missed the fact that there had to be sufficiently many *linearly independent* equations, not just equations. By considering the rank of the corresponding matrix, this can be shown easily.

Each subscriber has $\log \ell$ keys, which are on the path between a leaf node and the child node of the root. The colluders x_1 and x_2 have their keys,

$$\{(x_1, i, A_i(x_1), B(x_1)) | i \in V(x_1)\},$$
$$\{(x_2, j, A_j(x_2), B(x_2)) | j \in V(x_2)\}.$$

To apply our attack, we assume that the pirate decoder computes all the possible combinations of $K_{i,j}$ into the pirate key as follows.

$$K_p = \{K_{i,j} | i \in V(x_1), j \in V(x_2)\}.$$

However, only some parts are necessary. For example, in Fig. 1(a), if $x_1 \in \mathcal{U}_2$ and $x_2 \in \mathcal{U}_5$ then they need only 4 keys of $\{K_{2,13}, K_{9,13}, K_{12,5}, K_{12,10}\}$, since the other combinations of i, j cannot be included in $V(H)$ at once.

Proposition 2. *For a given valid input, the pirate decoder with K_p of the above form can compute a session key with probability $\frac{1}{2}$.*

Proof. This is similar to the Proposition 1. For a given header H, let $V(H) \cap V(x_1) = i$ and $V(H) \cap V(x_2) = j$. Since $V(x_1) \cap V(x_2) = \varnothing$ implies $i \neq j$, the combined key for this case, $K_{i,j}$ can be obtained as follows,

$$K_{i,j} := (x_1 + x_2, x_1^2 + x_2^2, \ldots, x_1^{2k-1} + x_2^{2k-1}, x_1^i, x_2^j, A_i(x_1) + A_j(x_2), B(x_1), B(x_2)).$$

Note that

$$\begin{aligned} A_i(x_1) + A_j(x_2) = \sum_{\substack{t=0 \\ t \neq i,j}}^{2k-1} a_t(x_1 + x_2)^t + (c_i x_1^i + a_i x_2^i) + (a_j x_1^j + c_j x_2^j) \\ - \lambda_i B(x_1) - \lambda_j B(x_2). \end{aligned} \tag{8}$$

The pirate decoder computes

$$\begin{aligned} \gamma = {} & h_{i,i}^{x_1^i} h_{j,i}^{x_2^i} h_{i,j}^{x_1^j} h_{j,j}^{x_2^j} \\ = {} & s^{x_1^i + x_2^j} \cdot g^{c_i x_1^i r_i + a_i x_2^i r_j + a_j x_1^j r_i + c_j x_2^j r_j}. \end{aligned}$$

If $r_i = r_j$, then it can compute the session key by

$$\begin{aligned} & \left(\gamma \prod_{\substack{t=0 \\ t \neq i,j}}^{2k-1} h_{i,t}^{x_1^t + x_2^t} \Bigg/ \hat{h}_i^{A_i(x_1) + A_j(x_2)} \bar{h}_i^{B(x_1)} \bar{h}_j^{B(x_2)} \right)^{\frac{1}{x_1^i + x_2^j}} \\ = & \left(\gamma \prod_{\substack{t=0 \\ t \neq i,j}}^{2k-1} (g^{a_t r_i})^{x_1^t + x_2^t} \Bigg/ g^{(A_i(x_1) + A_j(x_2) + \lambda_i B(x_1) + \lambda_j B(x_2)) r_i} \right)^{\frac{1}{x_1^i + x_2^j}} = s. \end{aligned} \tag{9}$$

For valid inputs, since each r_i is chosen from $\{R_0, R_1\}$ at random, the probability that $r_i = r_j$ is $\frac{1}{2}$ for any $i \neq j$. □

Similarly to the Theorem 1, this pirate key cannot be traced with high probability. Note that we still consider two colluders x_1 and x_2 who are in the left and right child of the root, respectively.

Theorem 2. *For the pirate decoder with a pirate key K_p of the above form by two colluders, they cannot be traced with the probability $1 - \frac{1}{2k}$.*

Proof. It is similar to the proof of Theorem 1, but since the key which is used for decryption varies for each header H, the precise description is somewhat complicated. Let $V(x_1) \cap \mathcal{L}_T = i$ and $V(x_2) \cap \mathcal{L}_T = j$, then $i < j$. The probability Pr_t for each $1 \leq t \leq n$ becomes: Let $v := V(H) \cap V(x_1)$ and $w := V(H) \cap V(x_2)$.

1. If $\mathcal{X} \cap \mathcal{U}_i = \varnothing$ and $t_2 \neq 2k$, then $Pr_t = 1$.
   Given a header H for each $\mathcal{X}$, since $r_v = r_w = R_0$, it can decrypt the H using $K_{v,w}$.
2. If $\mathcal{X} \cap \mathcal{U}_i = \varnothing$ and $t_2 = 2k$, then $Pr_t = \frac{1}{2}$.
   For the nodes v, w, since r_v and r_w are randomly chosen from $\{R_0, R_1\}$, the probability of $r_v = r_w$ is $\frac{1}{2}$, thus $Pr_t = \frac{1}{2}$.
3. If $\varnothing \neq \mathcal{X} \cap \mathcal{U}_i \neq \mathcal{U}_i$ and $\mathcal{X} \cap \mathcal{U}_j = \varnothing$ then $Pr_t = 0$.
   Since $\mathcal{U}_i$ where i is the leaf node is being finely revoked, $r_i = R_1$ and $r_w = R_0$, thus $Pr_t = 0$.
4. If $\mathcal{X} \supset \mathcal{U}_i$ then $Pr_t = 0$.
   Since $h_{v,v} = g^{z_v}$, $Pr_t = 0$.

Note that their black-box tracing algorithm also doesn't work for this pirate decoder. But by checking the probability of zero, a tracer can know the $\mathcal{U}_i$ which includes x_1, but cannot know who a traitor is exactly. From this we can conclude that the colluders of the pirate decoder cannot be traced with the probability $1 - \frac{1}{2k}$. □

Similar to the case of [6], we can still raise the probability of decryption on the valid inputs to 1 by 3 colluders. But this collusion are also related to the vulnerabilities which are argued in the next Subsections and from them the decryption probability can be 1 only by 2 colluders, we don't need to describe such collusion attacks.

4.2 Extraction of Secret Values

In this Subsection, we consider another attack which is applicable to the first and third methods by two and three colluders, respectively. The vulnerabilities also come from the structural problem that a subscriber has several keys from similar polynomials.

More precisely, a subscriber receives $\log \ell$ keys, but each couple of the key generation polynomials have the same coefficient except for two or three terms. By subtracting the polynomials with each other, 2 or 3 colluders can compute some secret values which should be kept securely. Since the procedure of extracting the secret values is similar in the first and third methods, we describe the procedure only on the third method.

Consider that three colluders x_1, x_2 and x_3 are in the same subscriber set corresponding to a leaf node. They are commonly included in the $\log \ell$ sets. For each $\mathcal{U}_i$ and $\mathcal{U}_j$ s.t. $i, j \in V(x_1) = V(x_2) = V(x_3)$ and $i < j$, they can set up the following system of equations.

$$\begin{cases} A_i(x_1) - A_j(x_1) = (c_i - a_i)x_1^i + (a_j - c_j)x_1^j - (\lambda_i - \lambda_j)B(x_1) \\ A_i(x_2) - A_j(x_2) = (c_i - a_i)x_2^i + (a_j - c_j)x_2^j - (\lambda_i - \lambda_j)B(x_2) \\ A_i(x_3) - A_j(x_3) = (c_i - a_i)x_3^i + (a_j - c_j)x_3^j - (\lambda_i - \lambda_j)B(x_3) \end{cases} \tag{10}$$

By solving this system of equations, they can compute $c_i - a_i$ for all i in $V(x_1)$. For a valid input H and the node i s.t. $i = V(H) \cap V(x_1)$, if there is another node $t \in V(H)$ s.t. $r_i = r_t$, they can obtain the session key by computing

$$\frac{h_{i,i}/h_{t,i}}{\hat{h}_i^{c_i - a_i}} = \frac{s \cdot g^{c_i r_i}/g^{a_i r_t}}{(g^{r_i})^{c_i - a_i}} = s, \tag{11}$$

where $h_{i,i}, \hat{h}_i$ are from H_i and $h_{t,i}$ is from H_t.

Note that the probability that there is such an additional node $t \in V(H)$ is $1 - \frac{2}{2^{\log \ell + 1}} = 1 - \frac{1}{\ell}$. This is because that a broadcaster selects $\log \ell + 1$ nodes and each r_v for $v \in V(H)$ is chosen in $\{R_0, R_1\}$ at random. So, the pirate decoder with the following secret values

$$K_p = \{c_i - a_i | i \in V(x_1)\}$$

can compute a session key correctly with high probability.

For example, in Fig. 1(a), let $x_1, x_2, x_3 \in \mathcal{U}_0$ be colluders, they also belong to $\mathcal{U}_8, \mathcal{U}_{12}$, then $\{0, 8, 12 \in V(x_1) = V(x_2) = V(x_3)\}$, they can solve the above system of equations and obtain $c_i - a_i$ for $i \in \{0, 8, 12\}$.

In this attack, since these values do not contain any information of the traitors, the tracer cannot find the identities of the traitors.

4.3 Constructing a Non-traceable Key Using Partial Keys

Although a tree-based tracing scheme has many good properties, the approach of combining the scheme of [6] with a tree structure has a critical vulnerability. This defect is related to the requirement of the Sel procedure of Encryption phase, so that it is applicable to all methods.

In the black-box tracing algorithm, in order to identify a traitor, one leaf node should be selected for the fine revocation. But this requirement has a flaw that if the colluders make a pirate key only with their keys corresponding to non-leaf nodes, then they can evade from the tracing algorithm. Furthermore, to achieve the minimum ciphertext size, the scheme requires a special form of node selection in the Sel procedure. If we follow the original Sel procedure, a tracer must take two subscriber sets corresponding to sibling leaf nodes of size $2k$, one set of size of $4k$, ..., one set of size $\frac{n}{2}$ corresponding to a child of the root node.

Note that the nodes of $i_{2\ell-4}$ and $i_{2\ell-3}$ are left and right child of the root node. Consider two colluders $x_1 \in \mathcal{U}_{2\ell-4}$ and $x_2 \in \mathcal{U}_{2\ell-3}$, i.e. they are in the left and right side of the tree T, respectively. They construct a pirate key K_p by collecting their keys corresponding to the nodes of depth 1. Then, for a valid input H, since there should be one node of depth 1 in the $V(H)$, the pirate decoder can always decrypt it using one of two keys. Since there are $\frac{n}{2}$ subscribers in each $\mathcal{U}_{2\ell-4}$ or $\mathcal{U}_{2\ell-3}$, they can be (almost) perfectly untraceable.

This simple but powerful attack is possible since the Sel always select a node of depth 1. One way of evading from this attack is to increase the number of selected nodes in the Sel procedure at the cost of efficiency. But this trial also fails by the following observations.

Since a subscriber should be able to decrypt valid inputs all the time and the height of the tree T should not be 1, we can find essential requirements of the Sel procedure as follows.

$$\bigcup_{v \in V(H)} \mathcal{U}_v = \mathcal{U}, \quad V(H) \setminus \mathcal{L}_T \neq \varnothing. \tag{12}$$

We consider two colluders x_1 and x_2 who are not in the sibling leaf nodes. They construct a pirate key K_p by collecting all their keys corresponding to non-leaf nodes. For example, in Fig. 1(a), suppose that two colluders $x_1 \in \mathcal{U}_1, x_2 \in \mathcal{U}_2$ for the third method, then a pirate key can be

$$K_p = \{(x_1, 8, A_8(x_1), B(x_1)), (x_1, 12, A_{12}(x_1), B(x_1)), (x_2, 9, A_9(x_2), B(x_2))\}.$$

Proposition 3. *For any valid input H through the Sel procedure satisfying the conditions (12), the pirate decoder with K_p of the above form can always compute a session key.*

Proof. Note that if the decoder has a key corresponding to the node v included in $V(H)$ then it can decrypt the H. For a subscriber u, the first condition of (12) can be rewritten as $V(H) \cap V(u) \neq \varnothing$. If we denote the nodes corresponding to the K_p by $V(K_p)$, then it is $V(K_p) = (V(x_1) \cup V(x_2)) \setminus \mathcal{L}_T$. Therefore,

$$\begin{aligned} V(H) \cap V(K_p) = & \quad ((V(H) \cap V(x_1)) \cup (V(H) \cap V(x_1))) \setminus \mathcal{L}_T \\ & = (V(H) \cap V(x_1) \setminus \mathcal{L}_T) \cup (V(H) \cap V(x_2) \setminus \mathcal{L}_T). \end{aligned}$$

This becomes empty only when $V(H)$ intersects at leaves with $V(x_1)$ and $V(x_2)$ at once. But since the two leaf nodes corresponding x_1 and x_2 are not siblings and the leaf nodes of $V(H)$ should be siblings, this set cannot be empty. From this, it is straightforward to decrypt a valid input all the time with the K_p. □

Since the keys corresponding to leaves are removed from the K_p, the untraceable probability is at least $1 - \frac{1}{4k}$. From the point of view of the pirate decoder, it is better to construct the pirate key K_p using the keys corresponding to the nodes of small depths. The above proposition shows that for all the Sel procedures which satisfy the conditions of (12), it is possible to construct a pirate decoder which can decrypt valid inputs with probability 1 but cannot be traced with the probability at least $1 - \frac{1}{4k}$.

5 Conclusion

In 2004, Matsushita and Imai proposed an efficient k-resilient public-key black-box traitor tracing scheme against self-defensive pirates, and in 2006, the same authors proposed a hierarchical key assignment method to reduce the ciphertext size by combining a complete binary tree with the former scheme. In this paper, we showed that the proposed schemes were vulnerable to our attack and it was possible to make an untraceable pirate decoder. Their schemes were based on multiple polynomials and our attack used a combination between different polynomials. The latter scheme also had a structural problem that we could exploit to deduce different types of attacks.

Although this paper is concentrated on breaking these schemes, we would like to remark that their contributions are still significant. In their schemes, there are many features to be discussed more, which were not explained in detail even in the original papers. In fact, we started this research from studying the work of [6]. After all, we happened to know that there were many delicate problems and they resolved some of them by novel ideas. Including modifying their schemes to resist our attacks, there still remain some problems to be resolved. We believe that much understanding of their schemes will lead us to some significant advance in the public-key black-box traitor tracing area.

References

1. Boneh, D., Franklin, M.: An Efficient Public Key Traitor Tracing Scheme. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 338–353. Springer, Heidelberg (1999)
2. Boneh, D., Sahai, A., Waters, B.: Fully Collusion Resistant Traitor Tracing with Short Ciphertexts and Private Keys. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 573–592. Springer, Heidelberg (2006)
3. Chor, B., Fiat, A., Naor, M.: Tracing traitors. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 257–270. Springer, Heidelberg (1994)
4. Kiayias, A., Yung, M.: On Crafty Pirates and Foxy Tracers. In: Sander, T. (ed.) DRM 2001. LNCS, vol. 2320, pp. 450–465. Springer, Heidelberg (2002)
5. Kurosawa, K., Desmedt, Y.: Optimum Traitor Tracing and Asymmetric Scheme. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 145–157. Springer, Heidelberg (1998)
6. Matsushita, T., Imai, H.: A Public-Key Black-Box Traitor Tracing Scheme with Sublinear Ciphertext Size against Self-Defensive Pirates. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 260–275. Springer, Heidelberg (2004)
7. Matsushita, T., Imai, H.: Hierarchical Key Assignment for Black-Box Tracing with Efficient Ciphertext Size. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 92–111. Springer, Heidelberg (2006)
8. Naor, D., Naor, M., Lotspiech, J.: Revocation and Tracing Schemes for Stateless Receivers. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 41–62. Springer, Heidelberg (2001)
9. Stinson, D., Wei, R.: Key Preassigned Traceability Schemes for Broadcast Encryption. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 144–156. Springer, Heidelberg (1999)

Tracing and Revoking Pirate Rebroadcasts

Aggelos Kiayias[⋆] and Serdar Pehlivanoglu[⋆]

Computer Science and Engineering, University of Connecticut
Storrs, CT, USA
{aggelos,sep05009}@cse.uconn.edu

Abstract. All content distribution systems are vulnerable to the attack of rebroadcasting: in a pirate rebroadcast a pirate publishes the content in violation of the licensing agreement. This attack defeats any tracing mechanism that requires interaction with the pirate decoder for identifying compromised keys. Merely tracing pirate rebroadcasts is of little use and one should be also able to revoke the involved traitor keys. The only currently known scheme addressing this issue is implemented as part of the Advanced Access Content System (AACS) used in Blu-Ray and HD-DVD disks. In this paper we perform an analysis of this construction and we find it has serious limitations: the number of revocations is bound by the size of the receiver storage (for the actual parameters reported this is merely 85 keys).

We address the limitations of the state of the art (i) by formally modeling the problem of tracing and revoking pirate rebroadcasts and (ii) by presenting the first efficient constructions of tracing and revoking pirate rebroadcasts that are capable of performing tracing for *unlimited numbers* of traitors and revoking *unlimited numbers* of users. We present three instantiations of our framework: our first construction achieves a linear communication overhead in the number of revoked users and traitors and is capable of eliminating a pirate rebroadcast by any number of traitors in time that depends logarithmically in the number of users and polynomially on the number of revocations and traitors. Our second construction assumes a fixed bound on the number of traitors and improves the elimination time to depend only logarithmically on the number of revocations. Both of these constructions require merely a binary marking alphabet. Our third construction utilizes a larger marking alphabet and achieves even faster pirate rebroadcast elimination; our analysis improves the previously known bound for the same alphabet size due to Fiat and Tassa from Crypto'99 while offering revocation explicitly.

1 Introduction

In the broadcast encryption setting a center broadcasts content to a number N of receivers. The center wishes to utilize the broadcast medium in such a way so that it can revoke at will any subset of size R from the population of receivers for any transmission. This requirement makes it impossible to hand the

[⋆] Research partly supported by NSF Awards 0447808, 0831304, 0831306.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 253–271, 2009.

same key to all receivers. The two trivial solutions to the broadcast encryption problem exhibit the trade-off between the receiver storage requirement and the ciphertext length. In the first trivial solution, each receiver obtains a personal key and subsequently the center can use the broadcast medium to simulate a unicast by transmitting a (vector) ciphertext of length $N-R$. While this solution is optimal from the receiver storage point of view, it wastes a lot of bandwidth. In the second trivial solution the center assigns a different key for any subset of receivers and each receiver is handed the keys for all the subsets it belongs to. In this case the ciphertext has optimal length but each receiver is required to store 2^{N-1} keys which is an exponential blow-up. Since the introduction of the first non-trivial scheme in [13] a number of subsequent works gave solutions exhibiting improved trade-offs [11,16,17,28]. Notably, [29] introduced the subset-cover framework that enabled the first schemes with ciphertext length linear in the number of revoked users R that allow unlimited revocations.

While revocation is an operation of critical importance, it is not sufficient for a content distribution system. Malicious receivers may obtain access to their keys (e.g., by reverse engineering their decoders) and then leak their key material to a "pirate." The pirate subsequently can construct a decoder that employs all these keys. The adversarial receivers in this setting are called traitors. A traitor tracing scheme [7] is a scheme that was suggested to deal with this problem: in a traitor tracing scheme, the center possesses the capability to interact with a pirate decoder and recover the identities of the traitors. Presumably after identification such traitors can be revoked. A number of subsequent works in [4,9,12,22,23,24,26,27,30,32,34,35,36,38,39,40] designed improved traitor tracing schemes and related codes dealing with this problem which has also being called the "clone decoder attack".

Combining the two functionalities of tracing and revoking in a single system is not straightforward. This was identified in [31] and Naor, Naor and Lotspiech [29] introduced trace and revoke schemes that are capable of offering a combined functionality that can deal with the problem of disabling pirate decoders. Subsequent work in the subset-cover framework of [29] gave better constructions [2,15,18,19,41] while also limitations were discovered in the form of a type of attacks called pirate evolution in [25].

Employing Trace and Revoke Schemes in practice. It should be noted that all trace and revoke schemes rely on multiple encryptions of the same plaintext under different keys something that suggests that they are not suitable for direct encryption of large messages. Given that the intended application scenario is content distribution it is expected that an encryption mechanism would have to handle large messages. The way this is solved is by employing hybrid encryption: the trace and revoke scheme is used to encrypt a one-time content key K and subsequently the content is appended encrypted with the key K.

The Pirate Rebroadcast Attack. The scenario of pirate rebroadcast attacks in the context of traitor tracing was introduced by Fiat and Tassa, [14]. In this scenario, the traitors first decrypt the content by using their key material and then once it is in clear text form, they rebroadcast the content[1]. In the hybrid encryption setting described above the attack is even worse: they can simply publish the content key K thus avoiding bulky uploads to online storage systems or other distribution mechanisms for content. Clearly a trace and revoke scheme would be useless against a pirate rebroadcast attack: the center is entirely powerless in handling such an attacker as the output of the rebroadcast itself provides no information about the traitor keys.

A solution to this problem would be feasible by employing watermarking techniques (e.g., such as those of [10]) so that the content itself becomes varied over the user population. As before the trivial solution would be marking the content individually so that each user has its own copy. As it was the case for broadcast encryption this solution wastes too much bandwidth. There are essentially two techniques known in the literature for obtaining non-trivial solutions that relax the bandwidth requirement: one is dynamic traitor tracing [14] and the other is sequential traitor tracing [33,21]. The idea in both cases is similar: the center will induce a marking of content and by observing the feedback from the pirate rebroadcast it will identify the traitors (refer to figure 1 for an illustration). The two methods differ in the following way: in the former, after each transmission the center obtains the feedback and tries to localize the suspect list by reassigning the marks adaptively. The number of traitors is not known beforehand and the system adjusts itself after each feedback. In the latter setting, the assignment of marks to the variations is predetermined (hence the transmission mechanism is not adaptive to the feedback). Note that depending on the parameters used, it may take a number of transmissions till the system converges and identifies one traitor.

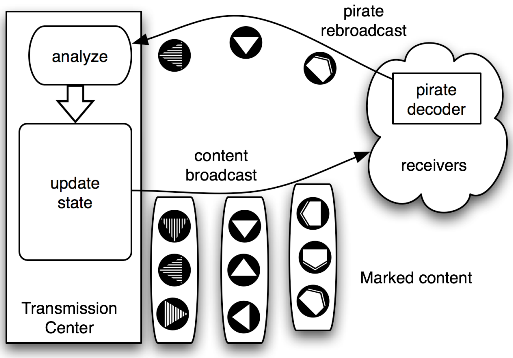


Fig. 1. The model for tracing a pirate rebroadcast attacker

[1] This attack has also being called the “anonymous attack” in [21,20]. We opt for calling it pirate rebroadcast instead given that we find it more descriptive of the adversarial action it describes; the same term was also used in the original paper [14]. Moreover, “anonymity” is a heavily overloaded term in the computer security literature and adding another connotation to it perhaps should be best if it is avoided.

Tracing and Revoking Pirate Rebroadcasts. In the above techniques that dealt with pirate rebroadcast attacks, the tracing mechanism does not provide a revocation capability. This is an important concern as it is not straightforward how to add revocation on top of either dynamic traitor tracing or sequential traitor tracing. To see why the straightforward approach fails suppose one decides to cascade a broadcast encryption at the decoder level with, say, a sequential traitor tracing scheme by composing the two encryption functions. This means that legitimate decoders will have to possess independent sets of keys from both schemes, i.e., one set of keys for the encryption/decryption involved in the sequential traitor tracing that binds the marked content to a receiver and one set of keys for the encryption/decryption involved in the broadcast encryption. It is easy to see that a pirate possessing the key material of as few as two traitor users can evade revocation at the decoder level by simply employing the keys of one user for decrypting the sequential traitor tracing layer and the keys of the other user for decrypting the broadcast encryption layer. In this attack scenario, the sequential traitor tracing scheme will successfully recover the identity of one of the traitors but subsequently revoking the recovered user will have absolutely no effect in the decryption capability of the pirate decoder (which will continue to operate due to the fact that it is using the broadcast encryption keys of the second uninidentified user).

It follows that any realistic solution for the problem of pirate rebroadcasts would require the design of a scheme that is capable of dealing with both tracing and revocation at the same time; failure of attaining these defensive properties simultaneously would make any scheme unusable in practice. The only known scheme offering both functionalities is implemented as part of the AACS [1] (and it also appears in [20]). The AACS is the current standard for content distribution that is employed in Blu-Ray and HD-DVD's. In this work we observe that the scheme employed in [1,20] has serious limitations (see below).

Our Results. In the present work we formally model the primitive of tracing and revoking pirate rebroadcasts and we present the **first constructions** that are capable of tracing and revoking an unlimited number of users. We also present a security and performance analysis of the only previously known scheme [20] and we find that the maximum number of revocations is bounded the receiver storage and the maximum traitor collusion that can be traced without false accusations is similarly bounded. Moreover, we provide a general design framework for tracing and revocation in the pirate rebroadcasting setting that permits a lot of flexibility in the choice of the basic parameters. The basic parameters of trace and revoking scheme for pirate rebroadasts is the *communication overhead* which is the amount of replication necessary in order to transmit a key, the *rebroadcast bound* which is the maximum number of transmissions a rebroadcasting pirate can "survive" before it is being entirely revoked in the system, and the *marking alphabet* which refers to the number of different variants of the content that the distribution center should create.

We present a number of instantiations of our design framework. Our first construction employs merely a binary marking alphabet and can withstand an unlimited number of traitors and revocations. The communication overhead is additively linear in the number of revoked users R and the number of traitors t. It follows that the communication overhead grows linearly in the number of malicious users; the constant hidden in the asymptotic notation is very small (it is $2R + 4t$ in the worst case). The pirate rebroadcast bound on the other hand is quadratic in the communication overhead and depends only logarithmically in the total number of users. This construction can thus keep on tracing and revoking arbitrary number of users with the only penalty of an extended rebroadcast bound as revocations accumulate. Our second construction also employs a binary alphabet and imposes a bound w on the size of the maximum traitor coalition. This enables us to improve the maximum pirate rebroadcast bound to depend on *logarithmically* in the number of revoked users (while being polynomially bounded on w). Finally our third construction improves further on the rebroadcast bound by employing a larger marking alphabet size of $2t + 1$ where t is the number of traitors; the bound is $O(t \log(N/t))$ and thus improves on the previously known convergence bound of $O(t \log N)$ for dynamic traitor tracing that was given by Fiat and Tassa [14] for the same alphabet size while offering revocation explicitly (while [14] offered tracing that relied on revocation without specifying how revocation is actually done).

Comparison with the AACS. The Advanced Access Content System (AACS) [1] is the current standard for content scrambling of Blu-Ray disks and HD-DVDs. It offers the functionality of a trace and revoke scheme (in fact it employs a variation of the subset-difference method of [29]) and it also offers a trace and revoking mechanism for pirate rebroadcasts (chapter 4 of [1], known as the "sequence key block") that was also detailed in [20]. While the subset-difference tracing and revocation mechanism is better understood much less attention has been given to the tracing and revocation for pirate rebroadcasts component of AACS. We perform an analysis based on the parameters suggested in [20] and we find that a maximum number of 85 revocations can be performed. Moreover, even if the parameters of [20] are variated to accomodate more revocations the scheme will suffer by a bound of the number of revocations that is limited by the number of stored keys in a receiver. Given that key assignment needs to be performed at system setup time this suggests that the approach taken in [20] has inherent limitations in terms of the number of revocations it allows. Moreover, the approach is also limited in terms of the maximum number of traitors that can be identified: the underlying traceability code suggested in [20] can identify at most 9 traitors. In fact, this is true even if a brute-force search is performed to try all possible coalitions (cf. section 4). Potentially this can be improved by a probabilistic analysis as suggested in [20] but in all cases once the number of traitors exceeds 9 the system has the potential to accuse innocent users. In contrast, the set of schemes we present here enjoy unlimited number of revocations and impose no bound on the number of traitors.

In our system, the device stores another layer of key system to process the key used to encrypt the watermarked content. In our setting, it is possible to have two devices with two different device keys, while at the same time, they obtain the same variant of the content. This has additional advantages as follows: In the AACS setting, the devices should update their key materials after 85 revocations. In many settings such update is impossible, e.g. in a DVD player case where the device is sold as an hardware in the market. In our system there is no need for such an update since a modified transmission block can account for all revocations needed.

Finally, it is possible to implement our construction in the AACS context using only the keys that are already present in the devices due to the implementation of NNL [29] without using any extra key material as it would be required for implementing the sequence key block method of [20].

2 Trace and Revoke Schemes for Pirate Rebroadcasts

A trace and revoke scheme $\mathtt{T}$ for pirate rebroadcasts consists of four procedures $\langle \mathsf{Init}, \mathsf{Transmit}, \mathsf{Receive}, \mathsf{Revoke} \rangle$. The parameters of the scheme are N, the number of users, and q the number of symbols in the transmission alphabet Σ. The scheme $\mathtt{T}$ is stateful and is parameterized by a set of states denoted by States. We define the four procedures below.

- Init. It is a probabilistic procedure that given 1^N it produces a set system $\langle \mathsf{N}, \mathcal{J}, \{\mathcal{I}_u\}_{u \in \mathsf{N}} \rangle$, where $\mathsf{N} = \{1, \ldots, N\}$, $\mathcal{J} \subseteq 2^{\mathsf{N}}$ and $\mathcal{I}_u \subseteq \mathcal{J}$ for all $u \in \mathsf{N}$ as well as an initial state $\sigma_0 \in \mathsf{States} \times 2^{\mathcal{J}}$.
- $\mathsf{Transmit}$. It is a probabilistic procedure that given a state $\sigma \in \mathsf{States} \times 2^{\mathcal{J}}$ and (optionally) a feedback symbol $f \in \{1, \ldots, q\}$ it produces a new state $\sigma' \in \mathsf{States} \times 2^{\mathcal{J}}$ and a subset of $\mathcal{J} \times \Sigma$.
- $\mathsf{Receive}$. It is a procedure that given $\mathcal{I}_u$ for some $u \in \mathsf{N}$ and a subset of $\mathcal{J} \times \Sigma$, it returns a symbol in Σ or $\perp$.
- Revoke. It is a procedure that given a state $\sigma \in \mathsf{States} \times 2^{\mathcal{J}}$ and a set $\mathsf{R} \subseteq \mathsf{N}$ it returns a new state $\sigma' \in \mathsf{States} \times 2^{\mathcal{J}}$.

Intuition. The four procedures in a trace and revoke scheme $\mathtt{T}$ for pirate rebroadcasts play the following role in an actual system instantiation. The Init procedure produces $\mathcal{J}$ which corresponds to the set of keys in the system and the sets $\mathcal{I}_u$ which determine the key assignment for each user u, i.e., each user $u \in \mathsf{N}$ will receive a set of keys corresponding to the set $\mathcal{I}_u$. It also produces the initial state $\sigma_0 = \langle \mathsf{state}_0, \mathcal{V}_0 \rangle$. The set $\mathcal{V}_0$ is the set of keys that are initially revoked in the system (typically $\mathcal{V}_0 = \emptyset$). The procedure $\mathsf{Transmit}$ possibly receives some feedback symbol $f \in \Sigma$ (originating from a pirate rebroadcast) and produces a subset J of $\mathcal{J} \times \Sigma$ that determines the way that the content should be transmitted. In particular for each $(j, s) \in J$ the system will transmit the encryption under the key $j \in \mathcal{J}$ of a version of the content marked with symbol $s \in \Sigma$. The $\mathsf{Receive}$ procedure will produce an accessible marking symbol given the content transmission and the keys of the user (note that in an actual implementation this procedure involves the identification of the watermark produced by a decoder). Finally Revoke updates the state of the system taking into account the set of revoked users R.

Next we define the correctness properties that are required from a trace and revoke scheme for pirate rebroadcasts.

Definition 1. ***Correctness.*** *For each of the four procedures of a trace and revoke scheme for pirate rebroadcasts we have a corresponding correctness property:*

Initialization Correctness. *For all* $\mathsf{R} \subseteq \mathsf{N}$ *it holds that there exist* $j_1, \ldots, j_v \in \mathcal{J}$ *for some* $v \in \mathbb{N}$ *such that* $\mathcal{I}_u \cap \{j_1, \ldots, j_v\} \neq \emptyset$ *for all* $u \in \mathsf{N} \setminus \mathsf{R}$.

Transmitting Correctness. *For any* $\sigma = \langle \mathsf{state}, \mathcal{V} \rangle$ *it holds that* $\mathsf{Transmit}(\sigma, f)$ *returns a state* $\sigma' = \langle \mathsf{state}', \mathcal{V}' \rangle$ *and a set* $J = \{(j_\ell, s_\ell)\}_{\ell=1}^{v} \in \mathcal{J} \times \Sigma$*, that satisfies the property* $\mathcal{I}_u \not\subseteq \mathcal{V}'$ *if and only if* $\exists j \in \mathcal{I}_u \setminus \mathcal{V}'$ *such that* $(j, s) \in J$ *for some* $s \in \Sigma$.

Receiving Correctness. *For any* $J = \{(j_\ell, o_\ell)\}_{\ell=1}^{v} \in \mathcal{J} \times \Sigma$*, it holds that* $\mathsf{Receive}$ $(\mathcal{I}_u, J)$ *picks any element of the set* $\{s \in \Sigma \mid \exists (j, s) \in J \text{ where } j \in \mathcal{I}_u\}$*, or returns* $\perp$ *if this set is empty.*

Revocation Correctness. *For any* $\sigma = \langle \mathsf{state}, \mathcal{V} \rangle$ *it holds that* $\mathsf{Revoke}(\sigma, \mathsf{R})$ *returns a state* $\sigma' = \langle \mathsf{state}', \mathcal{V}' \rangle$ *such that* $\mathcal{I}_u \subseteq \mathcal{V}'$ *for all* $u \in \mathsf{R}$.

Remarks on correctness. The correctness definition for the initialization property ensures that for any set of revoked users R it holds that we can find a set of keys $j_1, \ldots, j_v$ such that a user that is not revoked (i.e., $u \in \mathsf{N} \setminus \mathsf{R}$) has at least one key among $j_1, \ldots, j_v$. This requirement can be relaxed to hold only with very high probability over all possible choices for set systems on N users or it may be required to hold only for sets of revoked users that are bounded by some parameter r. Such relaxations may impact the system operation as they will hinder the exclusion of certain sets of users or introduce a failure probability in user revocation. Moreover, we note that the recovery of $j_1, \ldots, j_v$ given R should be done efficiently for a system to be useful in an applied setting.

The transmission correctness definition ensures that the subset J that is selected by $\mathsf{Trasmit}$ will enable a user that holds at least one unrevoked key to recover at least one symbol from the transmission alphabet. We note that an unrevoked user may recover many such symbols. The receiving correctness property specifies that the $\mathsf{Receive}$ function should choose one of the transmission symbols that the user can recover from a transmission or return $\perp$ in case that no symbol is accessible to the user. This would happen in case when all keys $\mathcal{I}_u$ of a certain user u have been included in the set $\mathcal{V}$ that holds the set of revoked keys in the state of the system. Finally, the revocation correctness given a set of users to be revoked it includes all their keys into the set of revoked keys $\mathcal{V}$.

Next we proceed to define the security aspects of a trace and revoke scheme against pirate rebroadcasts. We first define our notion of an adversary which is a pirate rebroadcast:

Definition 2. ***Pirate Rebroadcast.*** *A pirate rebroadcast of length* n *for a scheme* T *starting at state* σ_b *with respect to a set* $\mathcal{K} \subseteq \mathcal{J}$ *is a random variable* $\langle f_1, \ldots, f_n \rangle$ *over* Σ^n *that is subject to the constraint for all* $i = 1, \ldots, n$*, there exists* $(j, s) \in J$ *such that* $s = f_i$ *and* $j \in \mathcal{K}$ *where* (σ_{b+i}, J) *is distributed according to* $\mathsf{Transmit}(\sigma_{b+i-1}, f_{i-1})$ *and* $f_0 = \epsilon$.

In the above definition the set $\mathcal{K}$ is the set of keys that are available to the adversary. This may include the set $\mathcal{I}_u$ of all keys of a user u. A pirate rebroadcast consists of those symbols that are accessible to the adversary based on the way the system is choosing to transmit different symbols to subsets of users. Observe that any user in the system may produce pirate rebroadcasts of arbitrary length as long as its keys are not revoked (i.e., become part of the $\mathcal{V}$ set inside the system state).

Now we define the security property that needs to be satisfied by a trace and revoke scheme for pirate rebroadcasts. It states that any coalition of malicious users (or traitors) can only produce pirate rebroadcasts of a *bounded length* with high probability. This effectively means that the scheme is capable of identifying the source of a rebroadcast and over a number of transmissions eliminate it. The security property will be in the form of a bound μ that will specify the maximum number of transmissions a traitor coalition can withstand. The bound μ will be a function of the number of users N, the number of traitors t and the number of already revoked users R.

Definition 3. ***Traceability of Pirate Rebroadcasts.*** *We say that a scheme* $\mathtt{T}$ *satisfies* (μ, w)*-traceability against pirate rebroadcasts with probability* ϵ *provided that the following holds: for any set of* t *traitors* $\mathsf{T} \subseteq \mathsf{N}$ *with* $t \leq w$ *and any set of revoked users* R *it holds that, if* B *is the length of any pirate rebroadcast that starts at state* σ *with respect to the set of keys* $\mathcal{K} = \cup_{u \in \mathsf{T}} \mathcal{K}_u$*, then* $\mathbf{Pro}[B \leq \mu] \geq 1 - \epsilon$*. The probability distribution is taken over all states* σ *distributed according to* $\mathsf{Revoke}(\sigma_0, \mathsf{R})$ *where* σ_0 *is the initial state of the scheme as produced by* Init*. The parameter* μ *depends on* $N, t, R, \log(\frac{1}{\epsilon})$*.*

We define μ-full-traceability against pirate rebroadcasts when there is no bound w in the number of traitors. We remark that our tracing and revoking schemes for pirate rebroadcasts do not mandate the identification of the traitor users. This is because our aim is to eliminate any pirate rebroadcast that our system is given as feedback even if this rebroadcast does not uniquely identify a traitor. Observe that eventually, if the pirate keeps using the traitor key material available to it, all traitors will be revoked and hence they will be identified (but of course a pirate may stop rebroadcasting before that). This analysis approach is in line with the trace & revoke schemes of [29].

Communication Overhead. The communication overhead ψ of a scheme $\mathtt{T}$ is the amount of replication the scheme employs in order to trace the rebroadcasts. In order to measure the impact of both revocation and tracing on the communication overhead we will consider the case of a pirate rebroadcast generated by t users that occurs after the revocation of R users. In particular the communication overhead ψ of the scheme $\mathtt{T}$ will be a function of N, t, R that bounds from above the size of all sets J that are produced by $\mathsf{Transmit}$ following any pirate rebroadcast with respect to the set of the keys of any t traitors that start at state σ where $\sigma \leftarrow \mathsf{Revoke}(\sigma_0, \mathsf{R})$ and R is any set of R users.

3 Our Construction

Preliminaries. A subset cover scheme $SCS = (\mathsf{N}, \mathcal{J}, \mathsf{Cover}(\cdot), \mathsf{Split}(\cdot,\cdot))$ is a class of combinatorial design introduced in [29] that can be used for constructing key revocation methods. Note that $\mathsf{N} = \{1, \ldots, N\}$, $\mathcal{J} \subseteq 2^{\mathsf{N}}$. $Cover(\cdot)$ is a function that given a set of users $\mathsf{R} \subseteq \mathsf{N}$, it outputs a collection of subsets $\{\mathsf{S}_{i_1}, \ldots, \mathsf{S}_{i_v}\} \subseteq \mathcal{J}$, that is called a "broadcast pattern" or simply pattern and denoted by $\mathcal{P}$ such that $\mathsf{N} \setminus \mathsf{R} = \bigcup_{j=1}^{v} \mathsf{S}_{i_j}$. All subsets in $\mathsf{Cover}(\mathsf{N} \setminus \mathsf{R})$ are disjoint. $Split(\cdot,\cdot)$ is a function that given a broadcast pattern $\mathcal{P} = \{\mathsf{S}_{i_1}, \ldots, \mathsf{S}_{i_v}\}$ and a set of disjoint subsets $\mathcal{T} \subseteq \mathcal{J}$ it splits each subset of $\mathcal{P} \cap \mathcal{T}$ evenly (based on the "bifurcation property" of [29]) and returns an updated broadcast pattern that is derived from $\mathcal{P}$ by replacing the subsets $\mathcal{P} \cap \mathcal{T}$ with their splittings. If a subset in $\mathcal{P} \cap \mathcal{T}$ cannot be split it will simply be removed by $Split(\cdot,\cdot)$.

A fingerprinting code is a pair of algorithms $(\mathsf{CodeGen}, \mathsf{Tracing})$ that is defined as follows: $CodeGen$ is a probabilistic algorithm that is given input (n, ν, w, q) where $\nu = \log(\frac{1}{\epsilon})$ and ϵ is a security parameter, and it outputs a code $\mathcal{C}$ of n codewords over Σ^{ℓ} where $|\Sigma| = q$ (we refer to such codes as (ℓ, n, q)-codes) as well as a tracing key tk. $\mathsf{Tracing}$ is an algorithm that, informally, if c is constructed by a traitor coalition of size at most w by combining their codewords, it identifies at least one of the traitors with high probability. The fingerprinting code is called "open" if tk is empty.

The construction. We first describe our construction at a high level. In the initialization stage, we define the keys of all users based on a subset cover sheme. Upon detecting a pirate rebroadcast we will take advantage of the splitting property of the subset cover scheme and derive a pattern covering the active users that includes two or more subsets. Each subset in the pattern will be assigned potentially different symbols following a fingerprinting code that will be selected on the fly. After a sufficient number of transmissions (that matches the length of the code) the sequence of feedback symbols will form a pirate codeword and by applying the tracing algorithm of the fingerprinting code on it we will identify a subset that contains some traitors. This process will be repeated recursively until all the traitors are identified or the rebroadcast ceases. An illustration of this process is presented in figure 2.

We next describe our construction in more detail. We define the set of states of our scheme first: a state is a pair $(\mathsf{state}, \mathcal{V})$ such that (i) $\mathsf{state} \in \mathsf{States}$ consists of a pattern $\mathcal{P} \subseteq \mathcal{J}$ of keys, an instance of a fingerprinting code $(\mathsf{CodeGen}, \mathsf{Tracing})$ and a message transmission index m, and (ii) $\mathcal{V} \subseteq \mathcal{J}$ such that the following holds: u is such that $\mathcal{I}_u \not\subseteq \mathcal{V}$ if and only if $(\mathcal{I}_u \cap \mathcal{P}) \setminus \mathcal{V} \neq \emptyset$. Intuitively, $\mathcal{V}$ contains the keys of all revoked users and $\mathcal{P}$ is a set of disjoint subsets whose corresponding keys enable the transmission of content to the users who are not revoked.

– Init. Given 1^N, it produces a subset cover scheme $SCS = (\mathsf{N}, \mathcal{J}, \mathsf{Cover}(\cdot), \mathsf{Split}(\cdot,\cdot))$ which defines the set system $\langle \mathsf{N}, \mathcal{J}, \{\mathcal{I}_u\}_{u \in \mathsf{N}} \rangle$ by setting $\mathcal{I}_u$ to contain all $\mathsf{S} \in \mathcal{J}$ such that $u \in \mathsf{S}$. State $\sigma_0 = \langle \mathsf{state}_0, \mathcal{V}_0 \rangle$ is initialized as follows: $\mathcal{V}_0 = \emptyset$ and state_0 consists of the triple $(\mathcal{P}, FC, 0)$ that is selected as

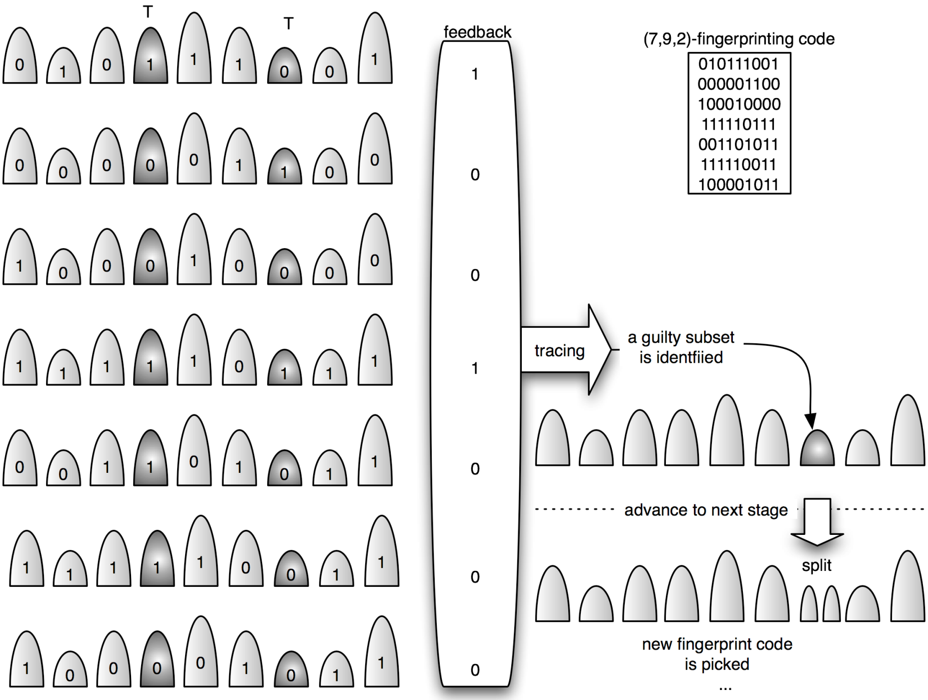


Fig. 2. Illustration of tracing and revoking a pirate rebroadcast with our construction

follows (i) $\mathcal{P} = \mathsf{Cover}(\mathsf{N})$, (ii) $FC \leftarrow \mathsf{CodeGen}(|\mathcal{P}|, \nu, w, q)$, i.e., $FC = (\mathcal{C}, tk)$ where $\mathcal{C}$ is a $(\ell, |\mathcal{P}|, q)$-code and tk is the corresponding tracing key. Note that each key index $\mathsf{S}_j \in \mathcal{P}$ is associated to a unique codeword $y_j \in \mathcal{C}$ for $j = 1, \ldots, |\mathcal{P}|$.

- $\mathsf{Transmit}$. Given the current state of the system $\sigma_{p-1} = \langle \mathtt{state}_{p-1}, \mathcal{V}_{p-1} \rangle$ and a feedback symbol $f \in \Sigma$, the system state is first updated to $\sigma_p = \langle \mathtt{state}_p, \mathcal{V}_p \rangle$. The update of the system is done as follows: the previous message transmission index m and the set of keys $\mathcal{P}$ are pulled out from σ_{p-1}. If $m < \ell$ where ℓ is the length of the code $\mathcal{C} = (\ell, |\mathcal{P}|, q)$ in state_{p-1} then m is increased by one and the feedback symbol f is stored. Otherwise (if $m = \ell$), we need to update the broadcast pattern $\mathcal{P}$. This is done as follows: the feedback values of all the ℓ recent transmissions are used to define a codeword $a \in \Sigma^\ell$ and then a set of subsets $T \subseteq \mathcal{P}$ is identified as follows: we compute $B = \mathsf{Tracing}(a, tk)$ and define T as $\mathsf{S}_j \in T$ iff $y_j \in B$ (here we use the 1-1 correspondence between the pattern subsets and codewords in $\mathcal{C}$). Then the broadcast pattern is updated by calling $\mathcal{P}' = \mathit{Split}(\mathcal{P}, T)$. A new fingerprint code $\mathcal{C}'$ should be sampled now to support as many codewords as the size of new broadcast pattern $\mathcal{P}'$ using $\mathsf{CodeGen}$ as described in the initialization step. The message transmission index m is set to 1. $\mathcal{V}_p$ is set to $\mathcal{V}_{p-1} \cup \{\mathcal{I}_u \mid \exists \mathsf{S}_j \in T \text{ where } \mathsf{S}_j = \{u\}\}$. This completes the description of the state update operation. After the state update the transmission function proceeds to select the set $J \subseteq \mathcal{J} \times \Sigma$.

This is done as follows: the triple $(\mathcal{P}, FC, m)$ is pulled out from σ_p. Then the subset J is defined to include all pairs $(\mathsf{S}_j, y_j[m])$ for $j = 1, 2, \ldots |\mathcal{P}|$ where $y_j[m]$ denotes the m-th symbol of the codeword $y_j \in \mathcal{C}$.

- $\mathsf{Receive}$. Given $\mathcal{I}_u$ for some $u \in \mathsf{N}$ and $J \subseteq \mathcal{J} \times \Sigma$, the procedure finds a pair $(j, s) \in J$ such that $j \in \mathcal{I}_u$ and returns s. If no such pair exists it returns $\perp$.
- Revoke. Given the current state $\sigma_{p-1} = \langle \mathsf{state}_{p-1}, \mathcal{V}_{p-1} \rangle$ and a set R, a new pattern $\mathcal{P}$ is selected as $\mathsf{Cover}(\mathsf{N} \setminus (\mathsf{R} \cup \mathsf{R}_{p-1})$ where $\mathsf{R}_{p-1} = \{u \mid \mathcal{I}_u \subseteq \mathcal{V}_{p-1}\}$. Subsequently, a new state state_p is formed by selecting a new fingerprinting code $FC \leftarrow \mathsf{CodeGen}(|\mathcal{P}|, \nu, w, q)$. The procedure returns $\langle \mathsf{state}_p, \mathcal{V}_p \rangle$ where $\mathsf{state}_p = (\mathcal{P}, FC, 0)$ and $\mathcal{V}_p = \mathcal{V}_{p-1} \cup (\cup_{u \in \mathsf{R}} \mathcal{I}_u)$.

Proposition 1. *The construction presented above satisfies correctness according to definition 1.*

Analysis of the construction. We next analyze the efficiency and security parameters of our construction. We first should examine in more depth the way the pattern updating algorithm $Split(\cdot, \cdot)$ operates. Without loss of generality we will instantiate our construction using for the underlying subset-cover scheme SCS the Subset-Difference method of [29]. The analysis is similar if another subset-cover scheme is being used such as those of [2,15,18,19,41]. We prove the following regarding the efficiency parameters of our construction:

Theorem 1. *The communication overhead ψ of our construction starting at state $\sigma \leftarrow \mathsf{Revoke}(\sigma_0, \mathsf{R})$ where σ_0 is the initial state of the scheme as produced by* Init *satisfies $\psi \leq 2|\mathsf{R}| + 4t$ where t is the number of traitors.*

For an illustration of how our construction works in combination with the subset-difference method we refer to figure 3 in the appendix. Next we present the analysis of the pirate rebroadcast bound for our construction.

Picking a Fingerprinting Code. The choice of the underlying fingerprinting code is flexible. It is possible to pick totally different codes in each stage (after a subset has been identified as containing a traitor) or keep the same code throughout. Moreover, this choice will be reflected in the deciphering process within the content transmission, hence the choice of fingerprinting code is independent from the keys stored in the device. The code is used to simply restructure the marking-assignment logically, by reassigning a subset to a new codeword.

A crucial difference regarding the selection of the fingerprinting code in our setting when compared to previous mechanisms that are employing some traceability or fingerprinting code, is that in our setting we only need codes with a number of codewords proportional to the number of revoked users and active traitors as opposed to the whole population. In contrast, previous works strived to produce codes with a number of codewords equal to the population size given a fixed small number of revoked users or traitors. Due to this important fact, we are able to employ fingerprinting codes that allow for arbitrary traitor collusions such as those presented in [6,40] without hurting the efficiency of our

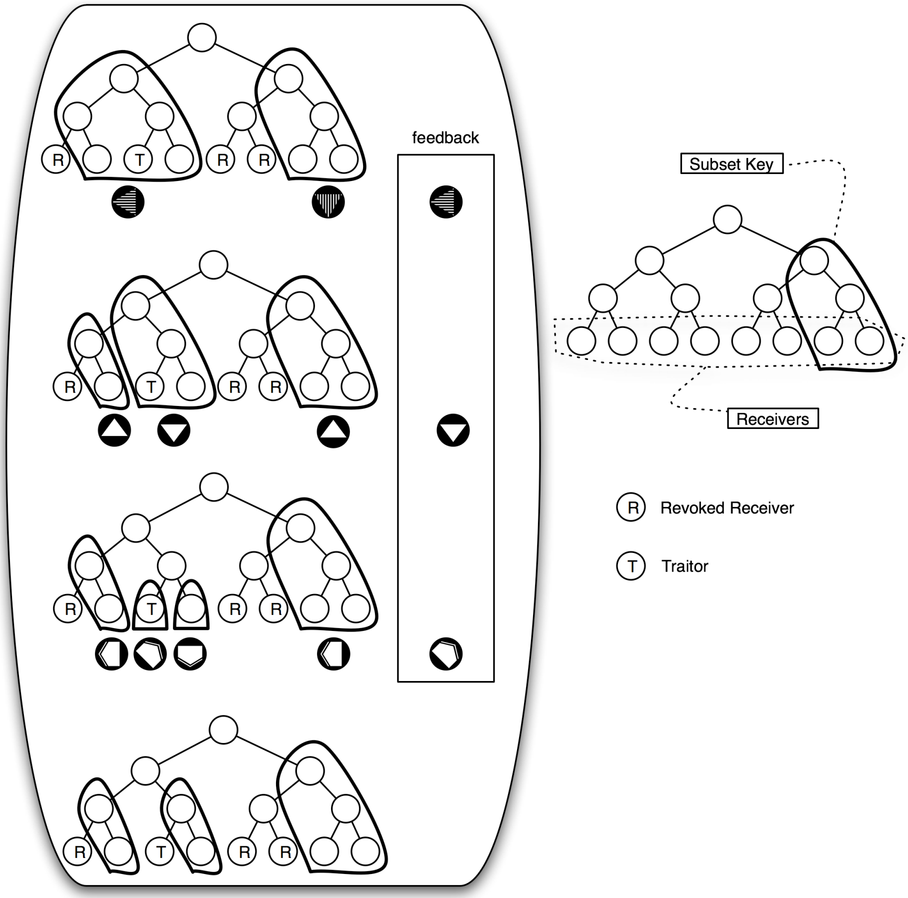


Fig. 3. Depiction of tracing a traitor following a pirate rebroadcast it produces using our construction (while employing the Subset-Difference method for key assignment)

construction and thus we can trace and revoke an unlimited number of traitors. We note that "picking a code" is not a computationally intensive operation as the codes we need can be sampled very efficiently or can be available in the form of a codebook and it is an operation that happens at the center and does not affect the complexity of the honest devices in any way.

Traceability of Pirate Rebroadcasts. The Tracing algorithm over the code $\mathcal{C}$ that is employed in the Transmit function will identify a subset containing a traitor with high probability. This is because of the fact that the codewords of $\mathcal{C}$ are assigned to subsets of devices, i.e., the detection of a "traitor" from the Tracing algorithm is now equivalent to finding a subset that contains a traitor. Once such a subset is found, this subset will be split into 2 subsets taking advantage of the *Split* function. The updated set of users, i.e the subsets in the new partition, will be reassigned new codewords from possibly a fresh fingerprinting code. Observe that the

$\mathsf{Tracing}$ algorithm of the fingerprinting code substitutes the "walking" argument that was employed in previous traitor tracing schemes (cf. [8,4,23,29]) that progressively randomized the pattern ciphertext till a position is identified that the pirate-box fails to decrypt successfully. This mechanism was the the basic procedure which tests a pirate box that is successful in decrypting with a given pattern by using some special tracing ciphertexts to output a subset containing a traitor. We note that such walking arguments cannot be used in our setting since they would require transmitting "garbage" to a subset of legitimate receivers, something that is unacceptable in the content distribution setting. This problem is not present in the previous works mentioned since they assume that they can analyze the pirate decoder in isolation from the transmission system (something impossible in the stronger adversarial setting we consider here).

We next analyze the convergence of our construction, i.e., the number of transmissions required to eliminate a pirate rebroadcast. At any system state σ_p with a set of revoked users R with $R = |\mathsf{R}|$ and a set of t subsets known to contain traitors, the number of subsets in the broadcast pattern covering the enabled set of users will be a function of R and t. Assuming that Subset Difference method of [29] is used, the size of broadcast pattern would be at most $2R + 4t$ as shown in Theorem 1. The pirate rebroadcast bound would depend on the tracing algorithm over the code $\mathcal{C}$ of size $O(R + t)$. We observe that it will require $O(\log N)$ stages to identify at least one traitor as this is the height of the Subset-Difference tree. Based on this we show the following:

Theorem 2. *Consider a set of traitors* T *with* $|\mathsf{T}| = t$*, a set of revoked users* R *with* $|\mathsf{R}| = R$*. If* σ *is a state distributed according to* $\mathsf{Revoke}(\sigma_0, \mathsf{R})$ *then the length of any pirate rebroadcast starting at* σ *is* $O(\ell \cdot t \cdot \log N)$ *with probability* $1 - t \cdot \log N \cdot \epsilon'$ *where* ℓ *is the length of the fingerprinting code used to instantiate the scheme and* ϵ' *the failure probability of the associated* $\mathsf{Tracing}$ *algorithm.*

Note that the dependency of μ in R is through the fingerprinting code length ℓ. Moreover, if there is a bound w on the number of traitors (i.e., $t \leq w$) this parameter will also appear as a function of ℓ. The proof is straight forward, length ℓ is required to detect at least one subset that contains a traitor. Identification of a single traitor requires at most $\log N$ new code selections. Identification of all traitors will then yield the bound given in the theorem. The actual pirate rebroadcast bound μ will depend on the choice of the code.

(**Instantiation 1**). In our first instantiation of the framework we use the optimal codes of [40] in conjunction to the subset-difference subset cover set system of [29]. This provides for a communication overhead $\psi = O(R+t)$ due to theorem 1 and a rebroadcast bound $\mu = O(t(R + t)^2 \log N \cdot \log((R + t)t \log N\epsilon^{-1}))$, where R is the number of revoked users, t the number of traitors, N the number of users and ϵ the error probability. The bound follows from theorem 2 and the fact that the length of Tardos' codes is $O(n^2 \log(n/\epsilon))$ where n is the number of codewords (that in our setting matches the communication overhead). Note that this scheme tolerates an *unlimited* number of traitors and revocations.

(**Instantiation 2**). Our second instantiation employs again Tardos' codes but assuming an upper bound on the number of traitors w. This provides for code length of $O(w^2 \log n/\epsilon)$ and given that in our setting we have that n is the number of codewords that should be equal to the communication overhead using theorem 2 we obtain a rebroadcast bound of $O(tw^2(\log N)\log((R+t)t\log N\epsilon^{-1})))$, i.e., with only logarithmic dependency on the number of revocations in the system.

(**Instantiation 3**). In our third instantiation we will take advantage of an increasing marking alphabet (instead of binary as in the previous two constructions). This will enable a very short rebroadcast bound of $O(t\log(N/t))$. We will use the complete subtree method of [29] to instantiate the subset cover system (that is superior for our purpose compared to the subset difference method). Recall that in the complete subtree method users are aligned as the leaves of a complete binary tree and the set system defines a key for any complete binary subtree of the total tree. Instead of relying on a fixed fingerprinting code to perform tracing as in the previous two constructions we will take advantage of our larger alphabet of $2t+1$ where t is the number of traitors to assign different versions to all subsets that result from the $Split(\cdot,\cdot)$ of the underlying subset cover scheme. We observe that there are at most $2t$ subsets that are formed after splitting in any pattern at any step of the tracing process. This means that we can use a $2t+1$ symbol alphabet. Given that the number of steps required to trace all t traitors equals the number of nodes in the Steiner tree of the t leaves that correspond to the traitors we conclude that the maximum pirate rebroadcast length is $O(t\cdot\log(N/t))$. This analysis not only improves on the previously known construction of Fiat and Tassa [14] that achieves $O(t\cdot\log N)$, but also it has an explicit description about how the revocation is performed.

4 Tracing Pirate Rebroadcasts in the AACS

In this section we describe[2] the part of AACS specifications [1] that deals with tracing pirate rebroadcasts (also published in [20]). We recast the description of that scheme in our terminology to facilitate the comparison with our framework. During the initialization of the system a sequence of partitions of the user population is formed. The partitions are determined according to the a fingerprinting code, specifically Reed-Solomon Code in the descriptions. After a sufficient number of transmissions (that matches the length of the code) the sequence of feedback symbols will form a pirate codeword and by applying the tracing algorithm of the fingerprinting code we will identify a traitor and revoke all of its keys. Note that revocation of some keys will effect the future feedbacks since the partitions remain unchanged and hence the traceability of the

[2] We note that this description may not necessarily reflect entirely all the details of the actual implementation of the AACS as many details are hidden, obfuscated or omitted in the references [1,20]. Still, we are confident that all the facts that we present about their construction are valid.

scheme will be harder in future transmissions which we will take advantage in our analysis later.

Formally, the construction of [20] is as follows: we define first a partition space $\mathsf{P}_{S,k}$ as a set of k disjoint subsets of S whose union yields the set S, i.e. $\{\mathsf{S}_1, \ldots, \mathsf{S}_k\} \in \mathsf{P}_{S,k}$ iff $\cup_{i=1}^{k} \mathsf{S}_k = S$ and $\mathsf{S}_i \cap \mathsf{S}_j = \emptyset$ for $i \neq j$. The state of the scheme $\mathsf{state} \in \mathsf{States}$ stores the history of feedback values from the pirate rebroadcast.

- Init. Given the set of devices $\mathsf{N} = \{1, 2, \ldots, 256^4\}$, it produces a set system $\langle \mathsf{N}, \mathcal{J}, \{\mathcal{I}_u\}_{u \in \mathsf{N}} \rangle$ where $\mathcal{J}$ consists of the subsets of 255 different partitions in $\mathsf{P}_{\mathsf{N},256}$. Denoting the j-th partition by v_j, the subsets of $\mathcal{J}$ are represented by $\mathsf{S}_{i,j}$ for $0 < i \leq 256$ and $0 < j < 256$ where $v_j = \{\mathsf{S}_{1,j}, \mathsf{S}_{2,j}, \ldots, \mathsf{S}_{256,j}\}$. The selection of partitions is done by using a Reed-Solomon code $\mathcal{C}$ with an alphabet $\Sigma = \{1, \ldots 256\}$ and of length 255. According to the the specifications, $\mathcal{C}$ is defined over polynomials of degree 3 in the finite field $\mathbb{F}_{256}$. Each receiver $u \in \mathsf{N}$ is assigned a codeword $y \in \mathcal{C}$ so that $\mathcal{I}_u$=$\{\mathsf{S}_{y[1],1}, \mathsf{S}_{y[2],2}, \ldots, \mathsf{S}_{y[255],255}\}$ where $y[j]$ denotes the j-th symbol of the codeword y. This will set $\mathsf{S}_{i,j} = \{u \in \mathsf{N} \mid u$ is assigned $y \in \mathcal{C}, y[j] = i\}$. $\sigma_0 = \langle \mathsf{state}_0, \mathcal{V}_0 \rangle$ is initialized such that both state_0 and $\mathcal{V}_0$ being emptyset.
- $\mathsf{Transmit}$. Given the current state of the system $\sigma_{p-1} = \langle \mathsf{state}_{p-1}, \mathcal{V}_{p-1} \rangle$ and a feedback value $f \in \{1, \ldots, 256\}$, the system state is first updated to $\sigma_p = \langle \mathtt{state}_p, \mathcal{V}_p \rangle$. The update of the state is done as follows: first state is updated to include feedback value f. If the sequence of feedback values stored in state_p makes it possible to identify a user $u \in \mathsf{N}$ as a traitor, then its keys are added to $\mathcal{V}_{p-1}$ to obtain $\mathcal{V}_p$, i.e. $\mathcal{V}_p = \mathcal{V}_{p-1} \cup \mathcal{I}_u$. The way the traitor detection is performed using the history of feedback values is explained in [20]. After state update, the transmission function proceeds to select the set $J \subseteq \mathcal{J} \times \Sigma$.

  This is done as follows: A set of partitions $\{v_{j_1}, \ldots v_{j_r}\} \subseteq \{1, \ldots, 255\}$ is chosen so that for any enabled device u, $(\mathcal{I}_u \setminus \mathcal{V}_p) \cap \{\mathsf{S}_{i,j_m} \mid 0 < m \leq r,$ $0 < i \leq 256\} \neq \emptyset$ holds. Note that the way the partitions are selected is not specified in [20] (presumably a heuristic can be used given that it is easy to test whether a given set of partitions satisfies the constraint or not[3]). We also remark that it is possible that such set of partitions does not exist, in which case the $\mathsf{Transmit}$ procedure will fail (i.e. the encryption phase put on top of $\mathsf{Transmit}$ procedure will exclude some honest devices from the transmission of actual content). We take advantage of this later in our analysis.

  The subset J is defined to include all pairs $(\mathsf{S}_{i,j}, i)$ where $\mathsf{S}_{i,j} \in \cup_{m=1}^{r} v_m \setminus \mathcal{V}_p$
- $\mathsf{Receive}$. Given $\mathcal{I}_u$ for some $u \in \mathsf{N}$ and $J \subseteq \mathcal{J} \times \Sigma$, the procedure finds a pair $(\mathsf{S}_{i,j}, s) \in J$ such that (1) $\mathsf{S}_{i,j} \in \mathcal{I}_u$ and (2) j is minimal among all subsets intersecting with $\mathcal{I}_u$ and the procedure returns s. If no such pair exists it returns $\perp$.

[3] Specifically in [20], it is stated(here columns refer to the partitions) "*However, after some number of columns depending on the actual number of compromised keys, the AACS licensing agency will know that only compromised devices would be getting the link key; all innocent would have found the output key in this column or in a previous column.*"

- Revoke. Given the current state $\sigma_{p-1} = \langle \mathsf{state}_{p-1}, \mathcal{V}_{p-1} \rangle$ the procedure returns $\langle \mathsf{state}_p, \mathcal{V}_p \rangle$ where $\mathsf{state}_p = \emptyset$ and $\mathcal{V}_p = \mathcal{V}_{p-1} \cup (\cup_{u \in \mathsf{R}} \mathcal{I}_u)$.

Plaintext Preprocessing and Marking. As seen above for a certain movie m the transmission center needs to produce 256 variations $\{m_1, \ldots, m_{256}\}$. A marking scheme of only 16 variations is being used though. For this reason a second (inner) Reed Solomon code $\mathcal{C}'$ is used over an alphabet $Q' = \{1, 2, \ldots 16\}$ that has length 15 and is defined over linear polynomials in the field $\mathbb{F}_{16}$. Thus, there are $16^2 = 256$ codewords in $\mathcal{C}'$, and each codeword in $\mathcal{C}'$ is a vector $\langle b_1, \ldots, b_{15} \rangle$. In order to transmit a movie m, the movie is split into 15 segments and the marking process is applied to each segment resulting in 16 different variations; let $s_{e,l}$ denote the e-th variation of the l-th segment, $l = 1, \ldots, 15$ and $e = 1, \ldots, 16$. Subsequently 256 versions of the movie are formed by employing the code $\mathcal{C}'$. In particular the i-th version of the movie m would be the string $s_{w_1,1} \ldots s_{w_{15},15}$ where $\langle w_1, \ldots, w_{15} \rangle$ is the i-th codeword of $\mathcal{C}'$. Note that this complicates somewhat the traceability analysis since the inner code is a 3-TA as opposed to a 9-TA that is employed for key assignment. Effectively this violates the marking assumption since it is possible for a coalition of size 4 to produce a movie for which the identification algorithm may not be able to identify any of its members. Without loss of generality we will ignore this issue for the moment and we will assume a “best case” behavior of their scheme for the sake of comparison. We refer to [20] for further discussion on this issue.

Revocation. The scheme of [20] as presented above has the capability to revoke a given set of users by selecting appropriately the set of columns that are used in the transmission. Here we observe that the revocation capability of the scheme is very limited. For the suggested parameters as described above we have the following:

Fact 1. *The scheme of [20] as presented above can support at most* 85 *revocations.*

To see why the above is true consider that key assignment is based on a Reed-Solomon code and during revocation a set of columns needs to be selected so that no innocent user is covered entirely by the set of keys assigned to the revoked users. It follows that a coalition capable of framing a user in the 9-TA code would cause the system to produce transmissions that disable some innocent users. It is easy to see that a coalition of 85 users may frame an innocent user in the system (this is because $3 \cdot 85 \geq 255$ where the coefficient 3 stems from the fact that this is the maximum number of locations that two codewords can agree in this code).

Traceability of Pirate rebroadcasts in AACS. In [20] it is argued based on a probabilistic analysis that a coalition of 9 traitors can be traced in 56 movie transmissions. We note that this analysis is performed in a setting without any revocations. Not only tracing after some number of revocations differs from the case when there is no revocation, it doesn't have a trivial solution. Morever, tracing after revocation will severely hurts the efficiency of the transmission overhead. On top of those limitations we observe the following on coalition bound:

Fact 2. *The scheme of [20] as presented may disable innocent users in case of a traitor coalition of a size larger than* 9 *occurs.*

To see why this is the case, consider that in the 9-TA Reed-Solomon code employed, it holds that there exist a coalition C_1 of size 10 that can produce a pirate codeword for which there exists an innocent user $u \notin C$ such that u's assigned codeword has maximum overlap with pirate codeword (strictly higher than any of the users in C_1). Note that any correct tracing algorithm would accuse the users that have very high overlap with the pirate codeword; hence no matter how the tracing algorithm of [20] operates the user u will be a likely outcome and hence this suggests that the revocation algorithm will be incapable of revoking the correct set of users (i.e., in some cases innocent users may be disabled from the system).

A Comparison of AACS with Our Constructions. Even if the above construction is variated over a different parameter selection and different codes the net effect would be the following: given the way the revocation algorithm works, the number of revocations will never exceed the length of the code employed (in fact they would be much less as illustrated above). Given that the code length selected in AACS [1] is 255 this suggests that the number of revocations feasible by this scheme is very limited. Even worse, as it is also pointed out in [20] as revocations accumulate the traceability of the underlying construction gets substantially reduced. In contrast our constructions enable an unlimited number of revocations against an unlimited number of traitors without incurring any degradation in security as the number of revocations accumulate. Moreover, given that the key assignment in our construction is based only on the subset-cover framework our tracing and revoking schemes for pirate rebroadcasts can be applied readily in the context of AACS (that employs a subset-cover key assignment already but used only for regular trace and revoking in the clone-decoder attack setting).

References

1. AACS Specifications specifications (2006), http://www.aacsla.com/
2. Attrapadung, N., Imai, H.: Graph-Decomposition-Based Frameworks for Subset-Cover Broadcast Encryption and Efficient Instantiations. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 100–120. Springer, Heidelberg (2005)
3. Berkman, O., Parnas, M., Sgall, J.: Efficient dynamic traitor tracing. In: SODA 2000, pp. 586–595 (2000)
4. Boneh, D., Franklin, M.: An Efficient Public-Key Traitor Tracing Scheme. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 338–353. Springer, Heidelberg (1999)
5. Boneh, D., Sahai, A., Waters, B.: Fully Collusion Resistant Traitor Tracing with Short Ciphertexts and Private Keys. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 573–592. Springer, Heidelberg (2006)
6. Boneh, D., Shaw, J.: Collusion-Secure Fingerprinting for Digital Data. IEEE Transactions on Information Theory 44(5), 1897–1905 (1998)

7. Chor, B., Fiat, A., Naor, M.: Tracing Traitors. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 257–270. Springer, Heidelberg (1994)
8. Chor, B., Fiat, A., Naor, M., Pinkas, B.: Tracing Traitors. IEEE Transactions on Information Theory 46(3), 893–910 (2000)
9. Chabanne, H., Hieu Phan, D., Pointcheval, D.: Public Traceability in Traitor Tracing Schemes. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 542–558. Springer, Heidelberg (2005)
10. Cox, I.J., Kilian, J., Leighton, F.T., Shamoon, T.: Secure spread spectrum watermarking for multimedia. IEEE Transactions on Image Processing 6(12), 1673–1687 (1997)
11. Dodis, Y., Fazio, N.: Public Key Broadcast Encryption for Stateless Receivers. In: Feigenbaum, J. (ed.) DRM 2002. LNCS, vol. 2696, pp. 61–80. Springer, Heidelberg (2003)
12. Dodis, Y., Fazio, N., Kiayias, A., Yung, M.: Scalable public-key tracing and revoking. In: PODC 2003, Proceedings of the Twenty-Second ACM Symposium on Principles of Distributed Computing (PODC 2003), Boston, Massachusetts, July 13-16 (2003)
13. Fiat, A., Naor, M.: Broadcast Encryption. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 480–491. Springer, Heidelberg (1994)
14. Fiat, A., Tassa, T.: Dynamic Traitor Tracing. Journal of Cryptology 4(3), 211–223 (2001)
15. Gentry, C., Ramzan, Z., Woodruff, D.P.: Explicit Exclusive Set Systems with Applications to Broadcast Encryption. In: FOCS 2006, pp. 27–38 (2006)
16. Gafni, E., Staddon, J., Lisa Yin, Y.: Efficient Methods for Integrating Traceability and Broadcast Encryption. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 372–387. Springer, Heidelberg (1999)
17. Garay, J.A., Staddon, J., Wool, A.: Long-Lived Broadcast Encryption. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 333–352. Springer, Heidelberg (2000)
18. Goodrich, M.T., Sun, J.Z., Tamassia, R.: Efficient Tree-Based Revocation in Groups of Low-State Devices. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 511–527. Springer, Heidelberg (2004)
19. Halevy, D., Shamir, A.: The LSD Broadcast Encryption Scheme. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 47–60. Springer, Heidelberg (2002)
20. Jin, H., Lotspiech, J.: Renewable Traitor Tracing: A Trace-Revoke-Trace System For Anonymous Attack. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 563–577. Springer, Heidelberg (2007)
21. Jin, H., Lotspiech, J., Nusser, S.: Traitor tracing for prerecorded and recordable media. In: Digital Rights Management Workshop, pp. 83–90 (2004)
22. Kiayias, A., Yung, M.: Self Protecting Pirates and Black-Box Traitor Tracing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 63–79. Springer, Heidelberg (2001)
23. Kiayias, A., Yung, M.: On Crafty Pirates and Foxy Tracers. In: Sander, T. (ed.) DRM 2001. LNCS, vol. 2320, pp. 22–39. Springer, Heidelberg (2002)
24. Kiayias, A., Yung, M.: Traitor tracing with constant transmission rate. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 450–465. Springer, Heidelberg (2002)
25. Kiayias, A., Pehlivanoglu, S.: Pirate Evolution: How to Make the Most of Your Traitor Keys. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 448–465. Springer, Heidelberg (2007)

26. Kurosawa, K., Desmedt, Y.: Optimum Traitor Tracing and Asymmetric Schemes. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 145–157. Springer, Heidelberg (1998)
27. Le, T.V., Burmester, M., Hu, J.: Short c-Secure Fingerprinting Codes. In: Boyd, C., Mao, W. (eds.) ISC 2003. LNCS, vol. 2851, pp. 422–427. Springer, Heidelberg (2003)
28. Micciancio, D., Panjwani, S.: Corrupting One vs. Corrupting Many: The Case of Broadcast and Multicast Encryption. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 70–82. Springer, Heidelberg (2006)
29. Naor, D., Naor, M., Lotspiech, J.B.: Revocation and Tracing Schemes for Stateless Receivers. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 41–62. Springer, Heidelberg (2001)
30. Naor, M., Pinkas, B.: Threshold Traitor Tracing. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 502–517. Springer, Heidelberg (1998)
31. Naor, M., Pinkas, B.: Efficient Trace and Revoke Schemes. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 1–20. Springer, Heidelberg (2001)
32. Hieu Phan, D., Safavi-Naini, R., Tonien, D.: Generic Construction of Hybrid Public Key Traitor Tracing with Full- Public-Traceability. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 49–63. Springer, Heidelberg (1996)
33. Safavi-Naini, R., Wang, Y.: Sequential Traitor Tracing. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 316–332. Springer, Heidelberg (2000)
34. Safavi-Naini, R., Wang, Y.: Collusion secure q-ary fingerprinting for perceptual content. In: Sander, T. (ed.) DRM 2001. LNCS, vol. 2320, pp. 57–75. Springer, Heidelberg (2002)
35. Safavi-Naini, R., Wang, Y.: New Results on Frameproof Codes and Traceability Schemes. IEEE Transactions on Information Theory 47(7), 3029–3033 (2001)
36. Safavi-Naini, R., Wang, Y.: Traitor Tracing for Shortened and Corrupted Fingerprints. In: Feigenbaum, J. (ed.) DRM 2002. LNCS, vol. 2696, pp. 81–100. Springer, Heidelberg (2003)
37. Silverberg, A., Staddon, J., Walker, J.L.: Efficient Traitor Tracing Algorithms Using List Decoding. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 175–192. Springer, Heidelberg (2001)
38. Staddon, J.N., Stinson, D.R., Wei, R.: Combinatorial Properties of Frameproof and Traceability Codes. IEEE Transactions on Information Theory 47(3), 1042–1049 (2001)
39. Stinson, D.R., Wei, R.: Combinatorial Properties and Constructions of Traceability Schemes and Frameproof Codes. SIAM Journal on Discrete Math. 11(1), 41–53 (1998)
40. Tardos, G.: Optimal probabilistic fingerprint codes. In: ACM 2003, pp. 116–125 (2003)
41. Jho, N., Hwang, J.Y., Hee Cheon, J., Hwan Kim, M., Hoon Lee, D., Sun Yoo, E.: One-Way Chain Based Broadcast Encryption Schemes. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 559–574. Springer, Heidelberg (2005)

Efficient Deniable Authentication for Signatures
Application to Machine-Readable Travel Document

Jean Monnerat[1,⋆], Sylvain Pasini[2,⋆⋆], and Serge Vaudenay[2]

[1] SwissSign AG, Zurich, Switzerland
[2] EPFL, Lausanne, Switzerland

Abstract. Releasing a classical digital signature faces to privacy issues. Indeed, there are cases where the prover needs to authenticate some data without making it possible for any malicious verifier to transfer the proof to anyone else. It is for instance the case for e-passports where the signature from the national authority authenticates personal data. To solve this problem, we can prove knowledge of a valid signature without revealing it. This proof should be non-transferable.

We first study deniability for signature verification. Deniability is essentially a weaker form of non-transferability. It holds as soon as the protocol is finished (it is often called offline non-transferability).

We introduce Offline Non-Transferable Authentication Protocol (ONTAP) and we show that it can be built by using a classical signature scheme and a deniable zero-knowledge proof of knowledge. For that reason, we use a generic transform for Σ-protocols.

Finally, we give examples to upgrade signature standards based on RSA or ElGamal into an ONTAP. Our examples are well-suited for implementation in e-passports.

1 Introduction

Digital signature schemes are one of the most important primitives in cryptography. A digital signature on a document allows to bind this document with a public key (e.g. an identity). One drawback is the privacy issue. Indeed, assume Alice signed a message and sent both the message and the signature to Bob. With a standard digital signature scheme Bob can verify its validity, but in addition he is able to convince anyone else of its validity. In some situations this transferability leads to privacy issues.

Monnerat, Vaudenay, and Vuagnoux [34,47,48] studied the e-passport standards and identified the privacy issue from leaking of signatures on private data. For instance, data such as official name, true date of birth, citizenship and a facial image together with a digital signature could easily be released on the Internet or sold by a malicious verifier, allowing to convince anybody of their

[⋆] Most of this work was done when the first author was at UCSD and supported by the Swiss National Foundation, PBEL2-116915.

[⋆⋆] Supported by the Swiss National Science Foundation, 200021-113329.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 272–291, 2009.

authenticity. None of these data is really confidential. For instance, the *true* date of birth of some person can fairly be estimated or propagated by gossiping. The person can still claim that the gossiped date of birth is incorrect and keep it private. What is more sensitive is a *proof* that a date of birth is true because the person can no longer deny evidence that the proof is correct. For this reason, the authors suggested to use non-transferable proof of signature knowledge. So, the passport can convince the border patrol of the authenticity of the data without revealing the signature.

In our scenario, we have a signer (the national authority), a prover (the e-passport), and a verifier (the border patrol). Clearly, the passport should not know the signing key which is kept secret by the national authority.

Full non-transferability requires a public-key infrastructure (PKI) for verifiers. Certificate verification should be secured to avoid transfer attacks using rogue keys. While a PKI for border patrols is proposed in the EAC standard [36], this is not enough for the privacy issue we have in mind. Indeed, the key of verifiers in EAC will only be checked for verifiers in a country with agreements with the home country. That is, non-transferability would only be enforced in friendly countries but not in others. This seems pretty weird.

For this reason, we want to avoid PKI for verifiers and we will focus on a weaker form of non-transferability, which holds after the protocol is complete. That is, we enforce *offline non-transferability* which is equivalent to deniability (sometimes called self-simulatability). Zero-knowledge proofs are inherently deniable in the plain model while zero-knowledge protocols in the common reference string (CRS) model or the random oracle model (ROM) are not necessarily deniable. However, protocols in the CRS or ROM are more attractive for efficiency reasons. So, we have to consider the notion of deniable zero-knowledge [40,41].

The above reasons motivated the study of non-transferable proof of signature knowledge with a strong focus on the efficiency. The goal is to find a protocol which can be implemented on e-passports for proving the knowledge of a valid signature to a border patrol in a setting where there is no PKI for border patrols and these latters may be dishonest. The international e-passport standard [35] proposes the use of RSA and ElGamal-based signatures schemes. The EAC [36] extension suggest that passports could run ECDH protocols. So, there is a little place for public-key cryptography.

Related work. Non-transitive signatures [16,38] and deniable message authentication [17] also deal with transferability issues but do not immediately apply for a three-party settings where an intermediate player (e.g. the e-passport) shows to another one that some data was authenticated by an authority.

Invisible signatures (aka undeniable signatures) were invented by Chaum and van Antwerpen [11] and they subsequently were studied in [4,9,15,23,33,37]. They are not universally verifiable, i.e. they make it impossible to tell valid and invalid signatures apart while making the signer able to prove validity or invalidity through an interactive protocol. Undeniable signatures only consider a two-party setting. One issue for our case is that in order to confirm or deny a signature, the prover must know the secret key.

With undeniable signatures, the signer cooperation is essential. Indeed, without its cooperation the signature is useless. The latter issue motivated the introduction of designated confirmer signatures [10]. In short, designated confirmer signatures work as undeniable signatures but the verification may be shifted from the signer to the confirmer. They were designed mostly to protect the verifier from signers who would refuse to participate in verification protocols.

Asokan, Shoup, and Waidner [1,2] proposed a solution to cross-exchange signatures between two parties in a fair way (using a trusted third party in a fair way). For that, they propose a way to transform a signature scheme into a verifiable escrow scheme. A verifiable escrow scheme is based on a homomorphism and allows to produce an escrow signature from a signature. Then, given the escrow signature, one can verify that it is really a signature without obtaining the signature. Finally, someone can recover the signature from the escrow signature by using the secret key. One drawback for our application is that the escrow signature is verifiable, thus it is some kind of signature and not deniable.

Non-transferability was studied by Jakobsson *et al.* [31,7] and they introduced designated-verifier proofs. The idea is to designate the signature to a verifier (using its public-key) and then only the designated-verifier can be convinced on the validity of the signature. One drawback is that the verifier must be known at the signature time. Later, Steinfeld *et al.* [46] introduced Universal Designated-Verifier Signature (UDVS). This scheme applies to a three party setting: signer, designator, and verifier. *Universal* refers to that any designator who obtained a universally verifiable signature from the signer is able to designate it to a verifier. This relies on a PKI for verifiers. The method for having every verifier attached to a public key is an overkill. This motivated Baek *et al.* [3] to define a weaker notion of non-transferability and they published the Universal Designated Verifier Signature Proof (UDVSP). The primitive is similar to the concept of UDVS except that no signature is given to the verifier. The designator does not need to know the verifier, only a signature proof is given to the verifier. This primitive assumes that verifiers are honest. Clearly, our application scenario does not meet this assumption and this construction does not remain secure when the verifiers may be malicious [32,45]. Indeed, considering malicious verifiers leads to transferable proofs by using the Fiat-Shamir transform [22]. In addition, The proposed UDVS or UDVSP constructions rely on bilinear mappings which seems not very easy to implement in the case of e-passports.

Recently, Shahandashti, Safavi-Naini, and Baek [45] worked on Credential Ownership Proofs (COP). There have some similar features as non-transferable signature. However, COP allow users to copy/share the credits which is clearly not desirable is the case of e-passports. They also protect against "double spending" which is not necessary in our case.

In this paper. To motivate our constructions, we start in Section 2 with a short overview on e-passports. In Section 3, we introduce some preliminaries and in particular, we recall the concept of deniable zero-knowledge in the CRS and RO models. In Section 4, we introduce the definition of an offline non-transferable authentication protocol (ONTAP). We propose a generic transform of a signature scheme into an ONTAP by using a deniable ZK proof of knowledge. In order to build secure ONTAP, we study strong construction of proofs of knowledge in Section 5. In particular, we study a generic transform of Σ-protocols. In Section 6, we propose ONTAP protocols which can be efficiently executed in constrained environments such as e-passports. In particular, we give an example based on the Guillou-Quisquater protocol for RSA-based signatures and another example based on the Schnorr protocol for ElGamal-based signatures.

Our work compared to others. As the UDVSP of Baek et al. [3] our ONTAP definition requires no PKI for verifiers. UDVSP and ONTAP are conceptually equivalent but the security notions differ. The UDVSP security from [3] uses three definitions, restrict to known message attacks and honest verifiers while we use two definitions, chosen message attacks and malicious verifiers. In addition to this, our instantiations of ONTAP can be built efficiently on standard signature schemes and need no change for the signing algorithm. UDVSP does not apply directly to RSA and ElGamal-based schemes while our proposed ONTAP implementations do. Our proposed implementations just require a modular exponentiation for the prover while the proposed UDVSP constructions require the use of bilinear mappings as well as signature transforms.

Recently, Shahandashti and Safavi-Naini [44] presented a construction for UDVS. As we saw before, UDVS requires PKI for verifiers and thus is not adapted in our case. However, they present a way for a signature holder to prove his signature knowledge to a verifier. They define a signature class $\mathbb{C}$ for which signatures can be converted in a public and a private part. The private part is simulatable and there exists a proof of knowledge for the private part. The signature holder simply needs to convert its signature, to send the public part to the verifier, and finally to prove his knowledge of the private part. They use this definition to designate a signature to a verifier by using a Fiat-Shamir transform on the interactive proof. Except the transform, we use a similar idea, i.e. a signature in two part, one simulatable and the other provable. The authors does not give any security proof (since it is not their main contribution). They use a classical Σ-protocol and thus their proof of knowledge is HVZK only. As seen before, HVZK is clearly not enough for the application we have in mind. Here we strengthen the knowledge proofs, we give formal security proofs and examples of implementations. Finally the scheme of [44] may loose deniability if a malicious verifier registers a rogue key.

2 Passive Authentication for MRTD

E-passports, formally called *machine-readable travel documents* (MRTD), are now available in many countries [35]. They use an embedded RFID chip to show evidence of a traveler identity through wireless communication.

The memory of the chip is organized in standard files: several *data groups* and one *security object document* (SOD). There are only two mandatory data groups: DG1 includes basic information such as the name of the person, its gender, date of birth, citizenship, as well as the passport number and validity; and DG2 contains a facial picture of the owner. The SOD includes the digest of each data group and a digital signature of this list of digests issued by the national authority. Clearly, the SOD gives evidence of someone's true name, or true age, or true gender, or true citizenship, etc. Providing the data groups and the SOD is called *passive authentication*.

Since all data is readable, chip cloning is possible. To solve this issue, there is an optional *active authentication* (AA) protocol. In that case, the chip possesses a pair public/private key. The public key is stored in a DG (and thus authenticated) while the private key is in a secure part of the memory (and so not clonable). Following AA, the reader simply sends a challenge and the chip signs it and give it back.

Due to the wireless access, data could be captured without the agreement by the holder. Following the standard, access to the chip can be protected using *basic access control* (BAC). In short, it proves to the chip that the reader have an optical access to the first page. It is not a real access control since anyone can implement an e-passport reader and read any passport without being authorized by public authorities.

BAC is by far insufficient since the new generation of e-passport will contain more private information such as fingerprint, address, etc. For this, the European Union is now promoting an *extended access control* (EAC) [36] which is based on more elaborate cryptographic protocols (semi-static ECDH key agreement with certificate) and terminal authentication based on a specific PKI. This PKI is also known to suffer from weaknesses (namely, the unreliable revocation procedure). In addition to this, EAC is only meant to protect non-mandatory data groups since mandatory ones should still be accessible to countries with no agreement to read extra information. This means that the SOD is not protected by EAC so will still leak evidence that a given protected data group is correct. Clearly, an adversary can still distinguish a correct EAC-protected data group from an incorrect one without being authorized to read it.

For this reasons, we propose to have the signature part of the SOD hidden and passive authentication replaced by some deniable authentication protocol. Note that the chip is able to carry out some RSA computation in AA. Therefore, we can assume that chips are able to run one or two RSA computations in the deniable authentication protocol.

3 Preliminaries

Let S be a finite set. We write $s \in_{\mathsf{u}} S$ to say that s is picked uniformly from S.

Throughout this article the term "algorithm" stands for a probabilistic polynomial-time (PPT) Turing machine modeled by deterministic functions in terms of an input and random coins.

We denote by $\mathsf{prot}_{\mathbf{P}(\alpha),\mathbf{V}(\beta)}(\gamma)$ an instance of the protocol "prot" between $\mathbf{P}$ and $\mathbf{V}$. The element γ denotes the common input of all participants, e.g. public keys, while α (resp. β) describes the private input of $\mathbf{P}$ (resp. $\mathbf{V}$). Note that when the protocol is not known or is implicitly known, the interaction between the two parties can be noted by $\langle \mathbf{P}(\alpha), \mathbf{V}(\beta)\rangle(\gamma)$.

In some cases, we need to only describe the view of $\mathcal{B}$ and we denote it by $\mathsf{View}_{\mathcal{B}}(\mathsf{prot}_{\mathcal{A},\mathcal{B}}(\cdot))$. We call "the view of $\mathcal{B}$" all inputs known by $\mathcal{B}$ (including the random tape and messages received by $\mathcal{B}$). All other messages can be computed from the view and the $\mathcal{B}$ algorithm.

Classical Digital Signature (DS) Schemes. We denote by $\mathcal{M}$ and $\mathcal{S}$ the message space and the signature space respectively. A (classical) *digital signature* (DS) scheme is defined by the three following algorithms: The $(K_p, K_s) \leftarrow \mathsf{setup}(1^\lambda)$ algorithm generates a key pair from a security parameter λ. The $\sigma \leftarrow \mathsf{sign}(K_s, m)$ algorithm outputs a signature $\sigma \in \mathcal{S}$ of a message $m \in \mathcal{M}$. The $b = \mathsf{verify}(K_p, m, \sigma)$ tells whether the pair (m, σ) is valid $(b = 1)$ or not$(b = 0)$.

The scheme is *complete* if for any $(K_p, K_s) \leftarrow \mathsf{setup}(1^\lambda)$, any message m, and any $\sigma \leftarrow \mathsf{sign}(K_s, m)$, then $\mathsf{verify}(K_p, m, \sigma) = 1$. The standard security requirement for a DS is the *existential unforgeability against a chosen-message attack* (EF-CMA) put forth by Goldwasser *et al.* [28]. This property ensures that nobody except the signer $\mathbf{S}$ can output a valid signature $\widehat{\sigma}$ for any new message m with a non-negligible probability.

Definition 1 (Security of DS). *Consider an adversary $\mathcal{A}$ against S. $\mathcal{A}$ plays a game against a challenger $\mathcal{C}$ who can sign messages. $\mathcal{A}$ is allowed to make queries to a signing oracle. The goal of $\mathcal{A}$ is to yield a valid pair $(\widehat{m}, \widehat{\sigma})$ such that $\widehat{m}$ was never sent to the signing oracle. The signature scheme is said* EF-CMA-secure *if no* PPT *adversary $\mathcal{A}$ can win this game with non-negligible probability.*

Proof of Knowledge and Deniable Zero-Knowledge. Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation with a polynomial-size witness, i.e., for any $(x, w) \in R$ we have $|w| \leq \mathsf{poly}(|x|)$. Let L_R be a language related to the binary relation R. L_R is the set of all x such that there exists a witness w and (x, w) is in R, i.e. $L_R = \{x : \exists w \text{ s.t. } (x, w) \in R\}$.

Let $(\mathbf{P}, \mathbf{V})$ be a pair of interactive Turing machines. For a given $x \in L_R$, $\mathbf{P}$ wants to prove to $\mathbf{V}$ that he knows the corresponding witness w. For this, $\mathbf{P}$ and $\mathbf{V}$ will use an *interactive proof* with common input x noted $\mathsf{proof}_{\mathbf{P}(w),\mathbf{V}}(x)$. At the end, $\mathbf{P}$ should have convinced $\mathbf{V}$ that he knows w. $\mathbf{V}$ outputs accept or reject. In order to formalize the notion of proof of knowledge we need to introduce the concept of *knowledge extractor* Ext. The Ext algorithm gets input x and access to the prover, while he attempts to compute w such that $(x, w) \in R$.

Consider any proof of knowledge between a prover $\mathbf{P}$ and a verifier $\mathbf{V}$. *Zero-knowledge* means that no information leaks to the verifier except the validity of the statement. This concept was formalized by Goldwasser, Micali, and Rackoff [26,27]. The main idea behind zero-knowledge is that any verifier should be able to run the simulator by himself (instead of interacting with a prover). However in the CRS model, the simulator is able to choose crs while no verifier is able to do that in reality. For this reason, we use the concept of *deniability* [40,41].

Definition 2 (Deniable Zero-Knowledge Proof of Knowledge). *Let* crs *be any common reference string (CRS). Let* H *be a random oracle. Let* $\kappa(x)$ *be a real valued function.* $\mathsf{proof}_{\mathbf{P}^{\mathsf{H}}(w),\mathbf{V}^{\mathsf{H}}}(x,\mathsf{crs})$ *is a* proof of knowledge *for the relation* R *with soundness error* $\kappa(x)$ *if the following holds :*

- Efficiency: $\mathbf{P}$ *and* $\mathbf{V}$ *are polynomially bounded.*
- Completeness: *On common input* x, crs*, if* $\mathbf{P}$ *has a witness* w *such that* $(x,w) \in R$*, then* $\langle \mathbf{P}^{\mathsf{H}}(w), \mathbf{V}^{\mathsf{H}} \rangle (x, \mathsf{crs})$ *always outputs* accept.
- Soundness: *Given a Turing machine* ${\mathbf{P}^*}^{\mathsf{H}}$, crs *and* H*, let* $\varepsilon(x)$ *be the probability that* $\langle {\mathbf{P}^*}^{\mathsf{H}}, \mathbf{V}^{\mathsf{H}} \rangle (x, \mathsf{crs})$ *accepts. There exists an extractor* Ext *and a constant* k *such that for any* crs*, any* H*, any* $\mathbf{P}^*$*, and any* x*, if* $\varepsilon(x) > \kappa(x)$*, then* $\mathsf{Ext}^{\mathbf{P}^*}(x)$ *outputs a witness* w *such that* $(x,w) \in R$ *within expected time* $\mathcal{O}\left(\frac{|x|^k}{\varepsilon(x)-\kappa(x)}\right)$ *where an access to* $\mathbf{P}^*$ *only counts as one step.*

A proof of knowledge $\mathsf{proof}_{\mathbf{P}^{\mathsf{H}},\mathbf{V}^{\mathsf{H}}}(\cdot)$ *for a relation* R *is* deniable zero-knowledge *if for any* PPT $\mathbf{V}^*$*, there exists a* PPT *simulator* $\mathsf{Sim}^{\mathsf{H}}$ *such that*

$$\{\mathsf{crs}, \mathsf{H}, \mathsf{View}_{\mathbf{V}^*}(\mathsf{proof}_{\mathbf{P}^{\mathsf{H}}(w),{\mathbf{V}^*}^{\mathsf{H}}(z)}(x,\mathsf{crs}))\}_{z\in\{0,1\}^*, x\in L_R} \text{ for arbitrary } w \in R(x)$$

and

$$\{\mathsf{crs}, \mathsf{H}, \mathsf{Sim}^{\mathsf{H}}(x,z,\mathsf{crs})\}_{z\in\{0,1\}^*, x\in L_R}$$

are computationally indistinguishable.

When crs and H are constant and H is polynomially computable, we obtain the definition in the standard model. When crs is constant, we obtain the random oracle model. When H is constant and polynomially computable, we obtain the CRS model. When $\mathbf{V}^*$ is restricted by $\mathbf{V}$, i.e. $\mathbf{V}^* = \mathbf{V}$, we obtain the honest verifier zero-knowledge (HVZK) definition.

It seems that the need for deniablity makes the CRS model collapse down to the plain model (see Pass [41]). Indeed, there exists an efficient generic transformation of deniable zero-knowledge protocols from the CRS model into the plain model. However this transformation adds some more rounds which increases the round complexity. So, deniable ZK protocols in the CRS model may still be attractive in practice.

Commitment Schemes. We define a keyed commitment scheme by the two following algorithms: The $(K_p, K_s) \leftarrow \mathsf{setup}(1^\lambda, R)$ algorithm generates a pair public/private key given random coins R. The $c = \mathsf{com}(K_p, m, r)$ algorithm allows to compute the commit value c for a given message m by using the public key K_p and random coins r. Knowing both c, m, and r (and K_p), the commitment is checked by $c = \mathsf{com}(K_p, m, r)$. A commitment scheme should be perfectly *hiding*, meaning that for any K_p generated by setup, $c \leftarrow \mathsf{com}(K_p, m, r)$ has a distribution which is independent from m. We assume that it is uniform. It should also be computationally *binding*, meaning that for any PPT algorithm given a random K_p generated by setup, the probability that it finds r,r',m,m' such that $m \neq m'$ and $\mathsf{commit}(K_p, m, r) = \mathsf{commit}(K_p, m', r')$ is negligible.

Trapdoor commitment schemes were introduced by Brassard, Chaum, and Crépeau [6]. A trapdoor commitment scheme is a keyed commitment scheme extend by a third algorithm, equiv, which defeats the binding property by using the secret key K_s. For any K_p, K_s generated by setup, any m, any $\widehat{c}$, and any execution $\widehat{r} \leftarrow \mathsf{equiv}(K_s, m, \widehat{c})$, we have $\widehat{c} = \mathsf{com}(K_p, m, \widehat{r})$.

For instance, a trapdoor commitment based on the discrete logarithm problem was proposed by Boyar and Kurtz [5]. Another trapdoor commitment scheme was proposed by Catalano et al. [8] based on the Paillier's trapdoor permutation [39].

Random Oracle Commitment Scheme. In the RO model, we can use the *RO commitment scheme*. Let H be a random oracle. The com algorithm with input m simply returns $c = \mathsf{H}(m\|r)$. To check the validity of the commit value c, given the message m' and the used random coins r', it is enough to check $c = \mathsf{H}(m'\|r')$.

4 Offline Non-transferable Authentication Protocol

Definition 3 (ONTAP). *We define an* offline non-transferable authentication protocol *(ONTAP) by the two following algorithms and the interactive verification protocol:*

- *The* $(K_p, K_s) \leftarrow \mathsf{setup}(1^\lambda)$ *algorithm generates a key pair.*
- *The* $\sigma = (\sigma_p, \sigma_s) \leftarrow \mathsf{sign}(K_s, m)$ *algorithm outputs a signature* $\sigma \in \mathcal{S}$ *of a message* $m \in \mathcal{M}$*.* σ *is split in a public part* σ_p *and a private part* σ_s*.*
- *The* $\mathsf{iProof}_{\mathbf{P}(\sigma_s),\mathbf{V}}(K_p, m, \sigma_p)$ *protocol allows a prover* $\mathbf{P}$ *to convince a verifier* $\mathbf{V}$ *that he knows a* σ_s *to complete* σ_p *in a valid signature for* m*. At the end* $\mathbf{V}$ *accepts or rejects.*

The scheme is complete *if for any* $(K_p, K_s) \leftarrow \mathsf{setup}(1^\lambda)$*, any message* m*, and any* $(\sigma_p, \sigma_s) \leftarrow \mathsf{sign}(K_s, m)$*,* $\mathbf{V}$ *always accepts in* $\mathsf{iProof}_{\mathbf{P}(\sigma_s),\mathbf{V}}(K_p, m, \sigma_p)$*.*

The UDVSP [3] uses a KeyGen algorithm which is equivalent to our setup algorithm. The Sign algorithm outputs a classical signatures universally verifiable by using the Verify algorithm, there is a $\mathsf{Transform}$ algorithm which generates a modified signature (with a public and secret part) from the universally verifiable one. Our sign algorithm may be built with the Sign and $\mathsf{Transform}$ algorithms from the UDVSP and conversely. We removed the Verify algorithm since it is useless with our definition. Finally, there is an interactive proof $\mathsf{IVerify}$ as our iProof. So, the two definitions are conceptually equivalent. The main difference comes from the security requirements.

The ONTAP is secure if it satisfies the next two definitions.

Definition 4 (Offline Non-Transferability of ONTAP). *Consider an adversary* $\mathcal{A}$ *against the ONTAP.* $\mathcal{A}$ *plays a game with a challenger* $\mathcal{C}$*. The goal of* $\mathcal{A}$ *is to get evidence that some message* $\widehat{m}$ *was signed. During the training phase,* $\mathcal{A}$ *is allowed to query a sign oracle denoted* Sign*. After the training phase,* $\mathcal{A}$ *selects some* $\widehat{m}$*,* $\mathcal{C}$ *signs it and reveals* $\widehat{\sigma}_p$*. Then,* $\mathcal{A}$ *runs a session of* $\mathsf{iProof}_{\mathbf{P}(\widehat{\sigma}_s),\mathcal{A}}(K_p, \widehat{m}, \widehat{\sigma}_p)$ *protocol. At the end of the game,* $\mathcal{A}$ *outputs all input*

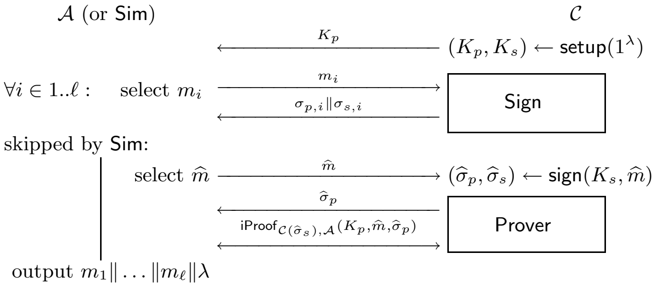

$\mathcal{A}$ (or Sim)
$\mathcal{C}$
K_p
$(K_p, K_s) \leftarrow \mathsf{setup}(1^\lambda)$
$\forall i \in 1..\ell$: select m_i
m_i
$\sigma_{p,i} \| \sigma_{s,i}$
Sign
skipped by Sim:
select $\widehat{m}$
$\widehat{m}$
$(\widehat{\sigma}_p, \widehat{\sigma}_s) \leftarrow \mathsf{sign}(K_s, \widehat{m})$
$\widehat{\sigma}_p$
$\mathsf{iProof}_{\mathcal{C}(\widehat{\sigma}_s),\mathcal{A}}(K_p, \widehat{m}, \widehat{\sigma}_p)$
Prover
output $m_1 \| \dots \| m_\ell \| \lambda$

Fig. 1. ONTAP Non-Transferable Game

queried to Sign *and its state* λ*. We introduce* Sim *which plays the same game but selects no* $\widehat{m}$ *and runs no* iProof *protocol.*

The ONTAP scheme is said offline non-transferable *if for any adversary* $\mathcal{A}$ *there exists a simulator* Sim *such that their output in the game of Fig. 1 are computationally indistinguishable.*

Definition 5 (Unforgeability of ONTAP). *Consider an adversary* $\mathcal{A}$ *against the ONTAP.* $\mathcal{A}$ *plays a game with a challenger* $\mathcal{C}$*. The goal of* $\mathcal{A}$ *is to convince* $\mathcal{C}$ *by running the* iProof *protocol that he knows* $\widehat{\sigma}_s$ *to complete* $\widehat{\sigma}_p$ *in a valid signature for* $\widehat{m}$*. During a training phase,* $\mathcal{A}$ *is allowed to query a sign oracle denoted* Sign*. On input message* m*,* Sign *answers the complete valid signature* (σ_p, σ_s)*. After this training phase,* $\mathcal{A}$ *selects a* $\widehat{m}$ *and a* $\widehat{\sigma}_p$ *with* $\widehat{m}$ *not sent to* Sign*.* $\mathcal{A}$ *simulates a prover to a honest verifier (see Fig. 2).*

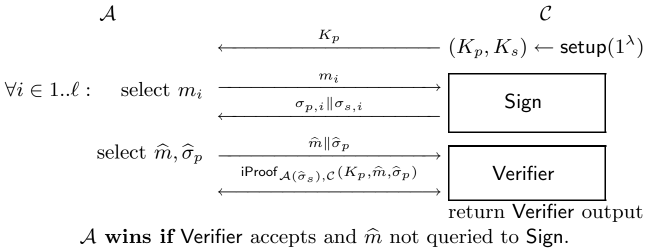

$\mathcal{A}$
$\mathcal{C}$
K_p
$(K_p, K_s) \leftarrow \mathsf{setup}(1^\lambda)$
$\forall i \in 1..\ell$: select m_i
m_i
$\sigma_{p,i} \| \sigma_{s,i}$
Sign
select $\widehat{m}, \widehat{\sigma}_p$
$\widehat{m} \| \widehat{\sigma}_p$
$\mathsf{iProof}_{\mathcal{A}(\widehat{\sigma}_s),\mathcal{C}}(K_p, \widehat{m}, \widehat{\sigma}_p)$
Verifier
return Verifier output
$\mathcal{A}$ **wins if** Verifier accepts and $\widehat{m}$ not queried to Sign.

Fig. 2. ONTAP Unforgeability Game

The ONTAP scheme is said unforgeable *if no* PPT *adversary* $\mathcal{A}$ *can make the honest verifier accepting the game of Fig. 2 with non-negligible probability.*

Clearly, Def. 5 implies unforgeability in the sense of Def. 1 since anyone able to forge a signature is also able to win the game of Fig. 2.

In Def. 5, we could give access to non-concurrent prover oracles to the adversary $\mathcal{A}^*$. Suppose it is the case and we denote them by Prover_j's. These oracles simulate the behaviour of an honest prover $\mathbf{P}$ in $\mathsf{iProof}_{\mathbf{P}(\sigma^*_{s,j}),\mathcal{A}^*}(K_p, m^*_j, \sigma^*_{p,j})$. Each oracle Prover_j is setup with a given message m^*_j and several iProof executions can be requested for the same m^*_j and some signature (σ_p, σ_s). Executions to the same Prover_j cannot be performed concurrently. This definition of unforgeability can be reduced to Def. 5 which uses no prover oracle. Suppose $\mathcal{A}^*$ is limited to m Prover oracles. We split $\mathcal{A}^*$ in several adversaries $\mathcal{A}_1^*$ to $\mathcal{A}_m^*$ playing modified games. Each $\mathcal{A}_i^*$ play with $\mathcal{C}$ where all Prover_j for $j \neq i$ are replaced by a query to the sign oracle and a simulation for the iProof protocol. So, only the Prover_i in the game with the adversary $\mathcal{A}_i$ uses a Prover oracle. Clearly, $\Pr[\mathcal{A}^* \text{ succeeds}] \leq \sum_{i=1}^{m} \Pr[\mathcal{A}_i^* \text{ succeeds}]$. Now we define adversaries A_i': each one plays the same game than $\mathcal{A}_i^*$ except than Prover_i is simulated by Sim as defined in Def. 4. By using the offline non-transferability property, the state of adversary $\mathcal{A}_i^*$ is computationally indistinguishable from the one of adversary $\mathcal{A}_i'$, so $\Pr[\mathcal{A}^* \text{ succeeds}] \leq \sum_{i=1}^{m} \Pr[\mathcal{A}_i' \text{ succeeds}] + \mathsf{negl}$. This proves that introducing a Prover oracle in Def. 5 does not strengthen our unforgeability notion when offline non-transferability is granted.

Theorem 6 (ONTAP construction). *Let $\mathcal{S}$ be a classical digital signature scheme in which the* sign *algorithm outputs a signature splittable in two parts: a public part σ_p and a private part σ_s. We assume there exists an algorithm* $\mathsf{simulate}$ *such that $\sigma_p \leftarrow \mathsf{simulate}(K_p, m)$ is computationally indistinguishable from the one generated by $\mathsf{sign}(K_s, m)$. Let* iProof *be a deniable zero-knowledge proof of knowledge for witness σ_s in the relation*

$$R(K_p \| m \| \sigma_p, \sigma_s) \Longleftrightarrow \mathsf{verify}(K_p, m, \sigma_p \| \sigma_s) \ .$$

If $\mathcal{S}$ is EF-CMA-secure, then the ONTAP $(\mathsf{setup}, \mathsf{sign}, \mathsf{iProof})$ *is secure.*

The required signature scheme $\mathcal{S}$ should be in the class $\mathbb{C}$ defined in [44] which includes many signature schemes. Note that there exists a Σ-protocol for any signature scheme since any NP relation has one [44]. However, such a protocol is in general not efficient. Thanks to the next section, we transform Σ-protocols into a denial ZK proof of knowledge.

Proof. We start with the constructed ONTAP scheme and consider the ONTAP security games. Assuming that $\mathcal{S}$ is EF-CMA-secure, the public signature is simulatable, and iProof is deniable ZK, we want to show that the ONTAP is unforgeable and offline non-transferable.

Unforgeability: We consider an adversary $\mathcal{A}$ playing the ONTAP unforgeability game with a challenger $\mathcal{C}$ as depicted on Fig. 2. $\mathcal{A}$ is bounded by a complexity T and is limited by ℓ queries to the oracle Sign. We split $\mathcal{A}$ in two parts: $\mathcal{A}_1$, which represents the three first moves of $\mathcal{A}$ on Fig. 2 and outputs a state λ, and $\mathcal{A}_2(\lambda)$, which represents the last two moves of $\mathcal{A}$. Thanks to the soundness, Ext fed with $\mathcal{A}_2(\lambda)$ produces $\widehat{\sigma}_s$ such that $\mathsf{verify}(K_p, \widehat{m}, \widehat{\sigma}_p, \widehat{\sigma}_s)$ holds. Hence, running $\lambda \leftarrow \mathcal{A}_1^{\mathsf{Sign}}$, then $\mathsf{Ext}^{\mathcal{A}_2(\lambda)}$ wins in the EF-CMA game which is not possible.

Offline Non-Transferability: We construct Sim by running $\mathcal{A}$ until $\widehat{m}$ is submitted. Then, Sim runs $\widehat{\sigma}_p' \leftarrow \mathsf{simulate}(K_p, \widehat{m})$ and continues to simulate $\mathcal{A}$ by feeding it with $\widehat{\sigma}_p'$. Clearly, $\mathcal{A}$ with the simulated $\widehat{\sigma}_p'$ reaches a state which is indistinguishable from $\mathcal{A}$ with a true signature $\widehat{\sigma}_p$. Then, we use the simulator for iProof to simulate the final state (and output) from $\mathcal{A}$. □

5 Deniable ZK from Σ-Protocols

The notion of Σ-protocol represents an important tool for the design of zero-knowledge protocols. Below, we first briefly recall the required material and refer to Damgård [14] for a detailed treatment. Then, we present a generic transform from any Σ-protocol into a deniable zero-knowledge proof of knowledge.

A Σ-protocol is a special 3-move honest-verifier zero-knowledge proof of knowledge for a relation R. We recall that for a pair $(x, w) \in R$, x is a common input for **P** and **V** and w is a private input for **P**. We usually denote the transcript (i.e., the three exchanged messages) by (a, e, z) and call the transcript "accepting" if an honest verifier **V** would accept the corresponding interactive proof execution. In Σ-protocols, e is a random bit-string which is (for the honest verifier) independent from a.

To fully characterize a Σ-protocol, we specify the algorithms which generate a and z, the domain of e, and the verifying algorithm executed by the verifier at the end. Let us denote them by PR_1, PR_2, $\{0,1\}^t$, and VER respectively. Finally, a Σ-protocol can be described formally as depicted on Fig. 3 where the notation ϖ_{P} (resp. ϖ_{V}) represents the random tape of the prover **P** (resp. verifier **V**).

$$\begin{tikzcd}[row sep=small]
\mathbf{P}(w;\varpi_{\mathrm{P}}) & (x) & \mathbf{V}(\cdot;\varpi_{\mathrm{V}}) \\
a = \mathsf{PR}_1(x,w;\varpi_{\mathrm{P}}) \arrow[rr, "a"] & & {} \\
{} & & e = \mathsf{trunc}_t(\varpi_{\mathrm{V}}) \arrow[ll, "e"'] \\
z = \mathsf{PR}_2(x,w,e;\varpi_{\mathrm{P}}) \arrow[rr, "z"] & & b = \mathsf{VER}(x,a,e,z)
\end{tikzcd}$$

Fig. 3. A Generic Σ-protocol

In addition to the above restrictions, a Σ-protocol must achieve efficiency and completeness following Def. 2, and must satisfy the two following conditions:
Special Soundness. For any $x \in L_R$ and any two accepting transcripts on input x, (a, e, z), (a, e', z') with $e \neq e'$, there exists a polynomial-time extractor $\mathsf{Ext}(x, a, e, e', z, z')$ which outputs a bit-string w such that $(x, w) \in R$.
Special HVZK. There exists a polynomial-time simulator Sim which for any x and a random e outputs a and z such that (a, e, z) has an identical probability distribution to the transcript generated by **P** and **V** on input x.

The special soundness (resp. special HVZK) guarantees that a Σ-protocol is sound (resp. HVZK). We define a weaker notion as follows.

Definition 7 ($\kappa(x)$-weak Σ-protocol). *Let κ be a real function. A* $\kappa(x)$-weak Σ-protocol *is a Σ-protocol with the special soundness property modified as follows:*

For any $x \in L_R$, any a, any $e \in \{0,1\}^t$, there exists a unique z such that $\mathsf{VER}(x, a, e, z) = 1$. *Denote* $z = \mathsf{Resp}(x, a, e)$.
There exists a polynomial-time algorithm Ext *such that for any $x \in L_R$, any a, and any $e \in \{0,1\}^t$, we have*

$$\Pr_{e' \in_{\mathrm{u}} \{0,1\}^t}[(x, \mathsf{Ext}(x, a, e, e', \mathsf{Resp}(x, a, e), \mathsf{Resp}(x, a, e'))) \in R] \geq 1 - \kappa(x)$$

Special soundness is achieved for $\kappa(x) = 2^{-t}$. $\kappa(x)$-weak Σ-protocols are sound with soundness error $\kappa(x)$. This comes from a simplified version of the proof of Th. 9 below.

Here we give two examples of $\kappa(x)$-weak Σ-protocols. The first example is the Guillou-Quisquater (GQ) protocol [30,29]. Let $N = pq$, e, and d be respectively an RSA modulus, the public and private exponents. For simplicity we assume that e is prime. (In practice, RSA keys use $e = 3$ or $e = 65537$). The GQ protocol allows to prove the knowledge of x such that $X = x^e \bmod N$, see Fig. 4a. The second example is from Schnorr [42,43]. Let g be the generator of a group G of prime order q. The Schnorr protocol allows to prove the knowledge of the discrete logarithm x in G of the element $X = g^x$, see Fig. 4b.

$$\begin{array}{rcl}
\mathbf{P}(x) & (N,e,X) & \mathbf{V} \\
\text{pick } y \in_{\mathsf{u}} \mathbb{Z}_N^* & & \\
Y = y^e & \xrightarrow{\quad Y \quad} & \\
 & \xleftarrow{\quad r \quad} & \text{pick } r \in_{\mathsf{u}} \{0,1\}^t \\
z = yx^r & \xrightarrow{\quad z \quad} & \text{check } z^e \overset{?}{=} YX^r
\end{array}$$

(a)

$$\begin{array}{rcl}
\mathbf{P}(x) & (g,q,X) & \mathbf{V} \\
\text{pick } y \in_{\mathsf{u}} \mathbb{Z}_q & & \\
Y = g^y & \xrightarrow{\quad Y \quad} & \\
 & \xleftarrow{\quad r \quad} & \text{pick } r \in_{\mathsf{u}} \{0,1\}^t \\
z = y + rx \bmod q & \xrightarrow{\quad z \quad} & \text{check } g^z \overset{?}{=} YX^r
\end{array}$$

(b)

Fig. 4. The GQ (a) and Schnorr (b) Protocols

Theorem 8. *Let t be the bit-length of the second move. The GQ protocol with prime exponent e is a $\frac{\lceil \frac{2^t}{e} \rceil}{2^t}$-weak Σ-protocol. The Schnorr protocol in a group of prime order is a 2^{-t}-weak Σ-protocol.*

Proof. The case of the Schnorr protocol is well known, so we concentrate on the GQ protocol.

Clearly we can define PR_1, PR_2, VER. Special HVZK is straightforward. Here we only need to prove that it is $\kappa(x)$-weak. Note that given the two first moves (Y,r), there exists a unique third move (z) for which $\mathbf{V}$ will accepts. It remains to prove the soundness and for that we should build an extractor Ext which outputs the witness given any $(I, Y, r_1, \mathsf{Resp}(I, Y, r_1))$ and a random $(r_2, \mathsf{Resp}(I, Y, r_2))$ with $I = (N, e, X)$.

Given (N, e, X), Y, r_1, r_2, z_1, z_2 such that $z_1^e = YX^{r_1} \pmod N$ and $z_2^e = YX^{r_2} \pmod N$ if $\gcd(r_1 - r_2, e) = 1$ we can find some integers a and b such that $ae + b(r_1 - r_2) = 1$ by using the Extended Euclid algorithm and then can compute $x = X^a z_1^b z_2^{-b} \bmod N$ which satisfies

$$x^e = X^{ae}(z_1^e)^b(z_2^e)^{-b} = X^{ae}(YX^{r_1})^b(YX^{r_2})^{-b} = X^{ae+b(r_1-r_2)} = X \pmod N$$

so a valid witness is extracted. Clearly, the GQ protocol is $\kappa(x)$-weak where $\kappa(x) = \max_{r_1} \Pr_{r_2}[\mathsf{gcd}(r_1 - r_2, e) \neq 1]$ and we find that $\kappa(x) \leq \frac{\lceil \frac{2^t}{e} \rceil}{2^t}$ since

$$\kappa(x) = \sum_{k=0}^{2^t-1} 1_{\mathsf{gcd}(r_1-k,e)\neq 1} \Pr[r_2 = k] = \frac{1}{2^t}\#\{\text{multiples of } e \text{ in } [r_1, r_1 + 2^t - 1]\}.$$

□

We transform a HVZK protocol into a deniable ZK protocol by adding a commitment step. This idea was proposed by Goldreich-Micali-Wigderson [25] and then reused by Goldreich-Kahan [24]. They prove that it is possible to achieve ZK in the standard model with a polynomial round complexity. Here, we want to prove that it is possible to achieve deniable ZK in the CRS or RO models with only 4 moves. At the same time, we achieve ZK in the standard model with one extra move. The extra move is necessary for sending the fresh public-key which replaces the common reference string. Note that Cramer-Damgård-MacKenzie [12] proposed a transform to achieve ZK but with a bigger round complexity while Damgård [13] proposed an efficient construction but without deniability. Clearly, for our application, deniability is mandatory in the CRS and RO models.

$$\begin{array}{rcl}
\mathbf{P}(w;\varpi_{\mathrm{P}}) & (x) & \mathbf{V}(\cdot;\varpi_{\mathrm{V}}) \\
R \leftarrow \mathsf{random}(\varpi_{\mathrm{P}}) & & \\
(K_p,K_s) \leftarrow \mathsf{setup}(1^\lambda,R) & \xrightarrow{\quad K_p \quad} & e \in_{\mathsf{u}} \{0,1\}^t,\ r \leftarrow \mathsf{random}(\varpi_{\mathrm{V}}) \\
 & \xleftarrow{\quad c \quad} & c = \mathsf{com}(K_p,e,r) \quad (\text{resp. } c = H(e,r)) \\
a = \mathsf{PR}_1(x,w;\varpi_{\mathrm{P}}) & \xrightarrow{\quad a \quad} & \\
\left(\text{resp. } c \overset{?}{=} H(e,r)\right) \quad c \overset{?}{=} \mathsf{com}(K_p,e,r) & \xleftarrow{\quad e\|r \quad} & \\
z = \mathsf{PR}_2(x,w,e;\varpi_{\mathrm{P}}) & \xrightarrow{\quad z\|R \quad} & (\widehat{K}_p,\widehat{K}_s) \leftarrow \mathsf{setup}(1^\lambda,R) \\
 & & \widehat{K}_p \overset{?}{=} K_p \\
 & & b = \mathsf{VER}(x,a,e,z)
\end{array}$$

Fig. 5. A Generic Transform of Σ-protocol

The protocol in the standard model is depicted on Fig. 5. Clearly, the prover should be ensured that nobody knows the trapdoor K_s. Consequently, the prover generates the key pair himself, he gives the public key on the first (extra) move and the private key on the last one.

Theorem 9. *Let $\mathcal{C}$ be a trapdoor commitment scheme, π be a $\kappa(x)$-weak Σ-protocol, and π' be its generic transform as depicted on Fig. 5.*

For any arbitrary large integer k, π' is a zero-knowledge proof of knowledge in the standard model with soundness error $\kappa'(x) = \max\left(\kappa(x), 1/|x|^k\right)$.

Clearly, if the trapdoor K_s is known by the verifier, the protocol remains honest-verifier zero-knowledge (this is essentially the Σ-protocol) but loses deniability. Indeed, a malicious verifier could take $e = \mathsf{OW}(a)$ and open c to e. The response z would become a transferable proof following the Fiat-Shamir paradigm [22] to transform interactive proofs into non-interactive ones. Thus, the K_p key should be trusted by the prover who believes that the verifier does not know the corresponding private key. In practice, a costless pragmatic solution could consist of using a hash function at the place of the commitment. This is essentially the instantiated variant with RO commitment scheme.

Theorem 10. *Let $\mathcal{C}$ be a trapdoor commitment scheme (resp. the RO commitment scheme). Let π be a $\kappa(x)$-weak Σ-protocol and let π' be its generic transform as depicted on Fig. 5 where K_p is the public key as setup in the commitment scheme (resp. where H is a random oracle).*

1. Ext' picks ϖ_{P} and set up $\mathbf{P}^*$ with ϖ_{P}.
2. Ext' plays the role of the verifier and runs a complete protocol with $\mathbf{P}^*$ who gives the trapdoor at the end. If this protocol does not fail (event A_1), this defines a first transcript $c, a, e_1\|r_1, z_1\|K_s$ such that $\mathsf{VER}(x, a_1, e_1, z_1, \varpi_p)$ outputs 1.
3. Ext' picks e_2 and computes $r_2 \leftarrow \mathit{equivocate}(K_s, c, e_2)$. Ext' runs another complete protocol with $\mathbf{P}^*$ set up with the same ϖ_p and uses messages c and $e_2\|r_2$. If this protocol does not fail (event A_2), this defines a second transcript $c, a, e_2\|r_2, z_2$ such that $\mathsf{VER}(x, a, e_2, z_2)$ outputs 1.
4. If one of the two protocols failed, Ext' aborts. Otherwise, using Ext with inputs (a, e_1, z_1) and (a, e_2, z_2), Ext' recovers w such that $(x, w) \in R$.

Fig. 6. The Knowledge Extractor Ext

For any arbitrary large integer k, π' is a deniable zero-knowledge proof of knowledge in the CRS model (resp. in the RO model) with soundness error $\kappa'(x) = \max\left(\kappa(x), 1/|x|^k\right)$.

Proof (Th. 9 and Th. 10). Efficiency and completeness of the protocols of Fig. 5 are trivial. So, we concentrate in proving the properties of soundness and deniable zero-knowledge.

Soundness. Let k be an arbitrary positive large integer and let $\mathbf{P}^*$ be any malicious prover which passes the protocol π' for x with an honest verifier $\mathbf{V}$ with probability $\varepsilon(x)$. Let $\kappa'(x)$ be the soundness error of the protocol π'. By Def. 2 it is assumed that $\varepsilon(x) > \kappa'(x)$.

Recall that for any $x \in L_R$ given two random accepting transcripts (a, e_1, z_1) and (a, e_2, z_2) there exists a polynomial-time extractor Ext which outputs the witness w such that $(x, w) \in R$ with probability $1 - \kappa(x)$ (over e_2).

We construct the extractor Ext' as described on Fig. 6. Thanks to the property of equiv, the extractor Ext' simulates perfectly a honest verifier for the malicious prover $\mathbf{P}^*$. Given ϖ_{P} and c, both protocols are independent and succeed with the same probability, let us denote it by $\Pr[A_j|\varpi_{\mathrm{P}}, c] = p_{\varpi_{\mathrm{P}},c}$ for $j = 1, 2$. The expected value of $p_{\varpi_{\mathrm{P}},c}$ over the random choice of ϖ_{P} and c is $\varepsilon(x)$. No matter whether A_j holds, let z_j be the unique z such that $\mathsf{VER}(x, a, e_j, z_j) = 1$. Let B the event that $\mathsf{Ext}(a, e_1, e_2, z_1, z_2)$ succeeds, i.e. $\Pr[\neg B] \leq \kappa(x)$. Furthermore, $\Pr[\neg B|A_1] \leq \kappa(x)$. We want to compute $\Pr[A_1 \wedge A_2 \wedge B]$ and we have :

$$\Pr[A_1 \wedge A_2 \wedge B|\varpi_{\mathrm{P}}, c] = \Pr[A_1 \wedge A_2|\varpi_{\mathrm{P}}, c] - \Pr[A_1 \wedge A_2 \wedge \neg B|\varpi_{\mathrm{P}}, c] \ .$$

Focusing on the right term, we have $\Pr[A_1 \wedge A_2 \wedge \neg B|\varpi_{\mathrm{P}}, c] \leq \Pr[A_1 \wedge \neg B|\varpi_{\mathrm{P}}, c] \leq p_{\varpi_{\mathrm{P}},c}\kappa(x)$, while the other term is $\Pr[A_1 \wedge A_2|\varpi_{\mathrm{P}}, c] = p^2_{\varpi_{\mathrm{P}},c}$, and we obtain $\Pr[A_1 \wedge A_2 \wedge B|\varpi_{\mathrm{P}}, c] \geq p_{\varpi_{\mathrm{P}},c}(p_{\varpi_{\mathrm{P}},c} - \kappa(x))$. Finally, we compute the expected value over the ϖ's and c's and using the Jensen's inequality on the function $x \mapsto x^2$, we obtain $\Pr[A_1 \wedge A_2 \wedge B] \geq \varepsilon(x)(\varepsilon(x) - \kappa(x))$. We conclude that the average number of running time of Ext' before it succeeds is $\frac{1/\varepsilon(x)}{\varepsilon(x)-\kappa(x)}$. Following Def. 2, it suffices to prove that $\frac{1/\varepsilon(x)}{\varepsilon(x)-\kappa(x)} \leq \frac{|x|^k}{\varepsilon(x)-\kappa'(x)}$ for any $\varepsilon(x) > \kappa'(x)$. It is the case when $\kappa'(x) = \max\left(\kappa(x), 1/|x|^k\right)$.

In the case of the CRS model, the extractor Ext' can be assumed to *know the trapdoor of the commitment.* In the standard model (Fig. 5), Ext' learns the

1. Sim launches $\mathbf{V}^*$ with a fresh random tape ϖ_{V} and receives c from $\mathbf{V}^*$.
2. Sim picks a random a using the same distribution than $\mathsf{PR}_1(\cdot)$. Thanks to the special HVZK property, this can be simulated by using Sim′ and obtaining (a, e_0, z_0). Sim then gives a to $\mathbf{V}^*$.
3. Sim receives e and r from $\mathbf{V}^*$ and checks $c = \mathsf{commit}(K_p, e, r)$.
   - If c is not valid, Sim stops the simulation and releases the transcript $(\varpi_{\mathrm{V}}, K_p, x, a)$.
   - Otherwise, i.e., if c is valid,
     (a) Sim rewinds $\mathbf{V}^*$ with the same random tape ϖ_{V} and receives the same c from $\mathbf{V}^*$ since ϖ_{V} is unchanged. Sim can thus guess that c will open to e.
     (b) Sim gives x and e to the simulator Sim′ described above in order to obtain a "good" transcript (a', e, z'). Sim sends a' to $\mathbf{V}^*$.
     (c) Sim receives e' and r' from $\mathbf{V}^*$ and checks $c = \mathsf{com}(K_p, e', r')$.
       - If c is not valid, Sim goes back to step 3a.
       - Otherwise, i.e., if c is valid
         i. If $e \neq e'$ (double opening of c), Sim aborts.
         ii. Sim finishes by yielding $(\varpi_{\mathrm{V}}, K_p, x, a', z')$ of the last interaction with $\mathbf{V}^*$.

Fig. 7. The Simulator Sim

trapdoor when event A_1 holds. In the case of the RO commitment, the proof is essentially the same: Ext′ creates two entries $\mathsf{H}(e_1, r_1) = \mathsf{H}(e_2, r_2) = c$ in the H table and executes both protocols by using only one entry. If by any chance $\mathbf{P}^*$ queries H with the other, the extraction fails. But this happens with negligible probability.

Deniable Zero-Knowledge. First, note that in the standard model deniable zero-knowledge (dZK) and zero-knowledge (ZK) are equivalent. So, in this proof we will show that all three protocols are dZK. This will imply that the protocol of Fig. 5 is ZK in the standard model.

We need to build a simulator able to simulate the interactions with any verifier $\mathbf{V}^*$ as described in Def. 2. Note that in the CRS model, the simulator Sim is not allowed to generate the common reference string. Let $K_p = \mathsf{crs}$ be any uniformly distributed random string. K_p is given to both, to Sim and to $\mathbf{V}^*$.

Recall that given any $x \in L_R$ and a random e there exists a polynomial-time simulator Sim′ which outputs a transcript (a, e, z) which has identical probability distribution than a transcript generated by $\mathbf{P}$ and $\mathbf{V}$ on input x.

We construct the simulator Sim as depicted on Fig. 7. Clearly, Sim always returns a complete protocol view from $\mathbf{V}^*$. It is either of type I $(\varpi_{\mathrm{V}}, K_p, x, a)$ or of type II $(\varpi_{\mathrm{V}}, K_p, x, a', z')$. The ϖ_{V} distribution is perfect as well as the view of type I. Let $A_{\varpi_{\mathrm{V}}, K_p, x}$ be the set of all a such that $\mathbf{V}^*(\varpi_{\mathrm{V}}, K_p, x, a)$ returns a valid c. The distribution of a' is the marginal distribution from PR_1 conditioned to set $A_{\varpi_{\mathrm{V}}, K_p, x}$. So, it is perfectly simulated as well. Finally, the unique z' is well simulated (the negligible probability of breaking the commitment has a negligible influence on the distribution) so we have a computationally indistinguishable simulator.

We still have to show that the average number of rewindings is polynomial. Let ϖ_{V} be a fix random tape of $\mathbf{V}^*$. Given $x \in L_R$, for w s.t. $(x, w) \in R$ we consider $\mathbf{V}^*_{\varpi_{\mathrm{V}}}$ interacting with $\mathbf{P}(x, w)$. We denote by $p_{\varpi_{\mathrm{V}}}(x)$ the probability that the commitment is incorrectly opened to $\mathbf{P}$. Since the distribution of a can be simulated, $p_{\varpi_{\mathrm{V}}}(x)$ does not depend on w. The number of executions of $\mathbf{P}$ is 1 with probability $1 - p_{\varpi_{\mathrm{V}}}(x)$ and $1 + \frac{1}{p_{\varpi_{\mathrm{V}}}(x)}$ with probability $p_{\varpi_{\mathrm{V}}}(x)$, so it is 2 on average. This proves that the simulator runs in expected polynomial time. □

6 ONTAP in Practice

6.1 ONTAP with Generic RSA Signature

We propose ONTAP-RSA: an ONTAP scheme which is generic for RSA-based signatures. It is based on a zero-knowledge variant of the GQ protocol.

Consider $h \leftarrow \mathsf{H}_{\mathsf{seed}}(m)$ be a formatting function and $b = \mathsf{V}(h, m)$ be a 0/1 check function returning 1 iff the formatted h is consistent with m.

Definition 11 (Generic RSA). *A generic RSA signature works in a group $\mathbb{Z}_N^*$ with $N \leftarrow pq$ and p, q are two $\frac{k}{2}$-bit random prime numbers. Let e, d such that $ed \equiv 1 \pmod{\varphi(N)}$ and e is prime (since several variant are commonly used we do not specify further the generation algorithm). The private key is $K_s \leftarrow (N, d)$ and the corresponding public key is $K_p \leftarrow (N, e)$*

The signature of message m consists of the tuple $\sigma = (\sigma_s, \sigma_p)$. The algorithm picks a random seed, computes the formatted message $\sigma_p \leftarrow \mathsf{H}_{\mathsf{seed}}(m)$, the signature $\sigma_s = \sigma_p^d \bmod N$ (using the private key K_s), and outputs σ_p and σ_s.

There exists a verification algorithm $\mathsf{verify}(K_p, m, \sigma_p, \sigma_s)$ which outputs 1 if the signature is valid, i.e. $\mathsf{V}(\sigma_p, m) = 1$ and $\sigma_s^e \bmod N = \sigma_p$, and 0 otherwise.

Clearly, the PKCS#1v1.5, ISO/IEC 9796, RSA-PSS standards all fit into this category.

The iProof protocol works as depicted on Fig. 8a. Note that K_p is the public-key related to the signature scheme while crs is the one related to the commitment scheme. The way to adapt to the plain model or random oracle model is straightforward.

Theorem 12. *Assume that the RSA-based signature is EF-CMA-secure and that the $\mathsf{com}(\cdot)$ is a trapdoor commitment scheme in the CRS model (resp. RO commitment scheme). The digital signature scheme added to the signature proof iProof of Fig. 8a forms a ONTAP scheme as defined in Def. 3 in the CRS model (resp. RO model). The soundness error is $\lceil \frac{2^t}{e} \rceil / 2^t$.*

With an extra round we obtain an ONTAP in the standard model (see Fig. 5).

Proof. Clearly, there exists an algorithm $\mathsf{simulate}(K_p, m)$ which outputs a σ_p computationally indistinguishable from the one generated by $\mathsf{sign}(K_s, m)$, i.e. $\sigma_p \leftarrow \mathsf{H}_{\mathsf{seed}}(m)$. Thanks to Th. 6, we only need to prove that the signature scheme is unforgeable and the protocol is deniable zero-knowledge. Unforgeability is already assumed. Efficiency and completeness of iProof are trivial. Soundness and deniable zero-knowledge properties of iProof are proven by Th. 8 and 9. □

6.2 ONTAP with Generic ElGamal Signature

Definition 13 (Generic ElGamal). *A* generic *ElGamal signature scheme works in a group $\mathbb{G}$ with a generator $g \in \mathbb{G}$ with order q. The private key is $K_s = x \in_{\mathsf{u}} \mathbb{Z}_q$ and the corresponding public key is $K_p = y = g^x$.*

$$\begin{array}{rcl}
\mathbf{P}(\sigma_s) & (\mathsf{crs}, K_p, m) & \mathbf{V} \\
\sigma_p = \sigma_s^e \bmod N & & \text{pick } \tilde{r} \\
 & & \text{pick } r \in_{\mathsf{u}} \{0,1\}^t \\
\text{pick } y \in_{\mathsf{u}} \mathbb{Z}_N^* & \xleftarrow{\quad c \quad} & c = \mathsf{com}(\mathsf{crs}, r, \tilde{r}) \\
Y = y^e \bmod N & \xrightarrow{\quad Y \| \sigma_p \quad} & \mathsf{V}(\sigma_p, m) \stackrel{?}{=} 1 \\
c \stackrel{?}{=} \mathsf{com}(\mathsf{crs}, r, \tilde{r}) & \xleftarrow{\quad r \| \tilde{r} \quad} & \\
z \leftarrow y\sigma_s^r \bmod N & \xrightarrow{\quad z \quad} & z^e \stackrel{?}{=} Y\sigma_p^r \pmod{N}
\end{array}$$

(a)

$$\begin{array}{rcl}
\mathbf{P}(\sigma_p, \sigma_s) & (\mathsf{crs}, K_p, m) & \mathbf{V} \\
 & & \text{pick } \tilde{r} \\
 & & \text{pick } r \in_{\mathsf{u}} \{0,1\}^t \\
\text{pick } \ell \in_{\mathsf{u}} [0, q-1] & \xleftarrow{\quad c \quad} & c = \mathsf{com}(\mathsf{crs}, r, \tilde{r}) \\
a = u^\ell \bmod p & \xrightarrow{\quad a \| \sigma_p \quad} & \mathsf{ver}(K_p, m, \sigma_p) \stackrel{?}{=} 1 \\
c \stackrel{?}{=} \mathsf{com}(\mathsf{crs}, r, \tilde{r}) & \xleftarrow{\quad r \| \tilde{r} \quad} & \sigma_p = (u, v, \xi) \\
z = \ell + r \cdot s \bmod q & \xrightarrow{\quad z \quad} & u^z \stackrel{?}{=} av^r \pmod{p}
\end{array}$$

(b)

Fig. 8. The iProof Protocols for ONTAP-RSA (a) and ONTAP-ElGamal (b)

The signature of a message m consists of the tuple $\sigma = (u, v, \xi, s) \leftarrow \mathsf{sign}(K_s, m)$. This tuple is split in two parts: a part $\sigma_p = (u, v, \xi)$ which can be perfectly simulated without K_s and a part $\sigma_s = s$.

There exists a verification algorithm $\mathsf{verify}(K_p, m, \sigma_p, \sigma_s)$ which outputs 1 if $u^s = v \ \wedge\ \mathsf{ver}(K_p, m, \sigma_p) = 1$ and 0 otherwise for some function ver.

ElGamal [21] (with a group of prime order), Schnorr [42,43], DSA [19,18], and ECDSA [20] signatures all meet the *generic* ElGamal requirements. Clearly, all of them respect the parameter and key generation. We give briefly four examples:

- The plain ElGamal signature is $u = g^k \bmod p$ for some random k, $v = g^{h(m)}y^{-u} \bmod p$, $\xi = \emptyset$, $s = \frac{h(m) - x\sigma_r}{k} \pmod{q}$ and the ver algorithm consists in checking $v \stackrel{?}{=} g^{h(m)}y^{-u} \pmod{p}$.
- The Schnorr signature is $u = g$, $v = g^s \bmod p$, $\xi = h(g^k \bmod p \| m)$, $s = k + x\xi \bmod q$ and the ver algorithm consists in checking $h(vy^{-\xi} \bmod p \| m) \stackrel{?}{=} \xi$
- The DSA signature is $u = g^k$ for some random k, $v = g^{h(m)}y^u$, $\xi = \emptyset$, $s = \frac{h(m)+xu}{k} \bmod q$ and the ver algorithm consists in checking $v \stackrel{?}{=} g^{h(m)}y^u \pmod{p}$.
- The ECDSA signature works over an elliptic curve with prime order n, with generator G, and with keys $K_s = d \in_{\mathsf{u}} [1, n-1]$ and $K_P = Q = dG$. The ECDSA signature is $u = (u_x, u_y) = kG$, $v = h(m)G + \bar{u}_x Q$, $\xi = \emptyset$, $s = \frac{h(m)+d\bar{u}_x}{k} \bmod n$ and the ver algorithm consists in checking $v \stackrel{?}{=} h(m)G + \bar{u}_x Q \pmod{n}$. (Here we use additive notations and the $u_x \mapsto \bar{x}_x$ mapping is defined in [20]).

In order to build a non-transferable signature, instead of revealing the private part of the signature, we will prove that we know it. Clearly, we will use a zero-knowledge proof as before. The required proof of knowledge should allow **P** to prove to **V** that he knows s such that $u^s = v$. Note that this is the proof of the knowledge of the discrete logarithm. The identification protocol from Schnorr [42,43] is a Σ-protocol when q is prime proving exactly that. Consequently, we applied our generic transform from Th. 9 and we obtain the verification protocol of Fig. 8b which is deniable zero-knowledge in the CRS model. We thus obtain several schemes: ONTAP-ElGamal, ONTAP-Schnorr, ONTAP-DSA, ONTAP-ECDSA, and so on.

Theorem 14. *Assume that the ElGamal-based signature is EF-CMA-secure and that the* com(·) *is a trapdoor commitment scheme in the CRS model (resp. RO commitment scheme). The digital signature scheme added to the signature proof* iProof *of Fig. 8b forms a ONTAP scheme in the CRS model (resp. RO model). The soundness error is* 2^{-t}.

See proof of Th. 12.

7 Conclusion

We studied the deniability notion in the case of digital signature verification. We proposed a new primitive called ONTAP as an offline non-transferable proof for holding a valid signature. As example, we presented an efficient signature proof for RSA-based and ElGamal-based signatures. Our protocol offers an adequate solution for private data authentication especially in the context of e-passports. It is compatible with all standard signature schemes.

References

1. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures (extended abstract). In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer, Heidelberg (1998)
2. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures (extended abstract). IEEE Journal on Selected Areas in Communications 18(4), 593–610 (2000)
3. Baek, J., Safavi-Naini, R., Susilo, W.: Universal Designated Verifier Signature Proof (or How to Efficiently Prove Knowledge of a Signature). In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 644–661. Springer, Heidelberg (2005)
4. Boyar, J.F., Chaum, D., Damgård, I., Pedersen, T.P.: Convertible Undeniable Signatures. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 189–205. Springer, Heidelberg (1991)
5. Boyar, J.F., Kurtz, S.A., Krentel, M.W.: A discrete logarithm implementation of perfect zero-knowledge blobs. Journal of Cryptology 2(2), 63–76 (1990)
6. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. Journal of Computer and System Sciences 37(2), 156–189 (1988)
7. Camenisch, J., Michels, M.: Confirmer Signature Schemes Secure against Adaptive Adversaries. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 243–258. Springer, Heidelberg (2000)
8. Catalano, D., Gennaro, R., Howgrave-Graham, N., Nguyen, P.Q.: Paillier's cryptosystem revisited. In: CCS 2001, pp. 206–214. ACM Press, New York (2001)
9. Chaum, D.: Zero-Knowledge Undeniable Signatures. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 458–464. Springer, Heidelberg (1991)
10. Chaum, D.: Designated Confirmer Signatures. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 86–91. Springer, Heidelberg (1995)
11. Chaum, D., van Antwerpen, H.: Undeniable Signatures. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 212–217. Springer, Heidelberg (1990)

12. Cramer, R., Damgård, I., MacKenzie, P.D.: Efficient Zero-Knowledge Proofs of Knowledge Without Intractability Assumptions. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 354–372. Springer, Heidelberg (2000)
13. Damgård, I.: Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 418–430. Springer, Heidelberg (2000)
14. Damgård, I.: On Σ-protocols. Lecture Notes (2005)
15. Damgård, I., Pedersen, T.P.: New Convertible Undeniable Signature Schemes. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 372–386. Springer, Heidelberg (1996)
16. Desmedt, Y.: Subliminal-Free Authentication and Signature (Extended Abstract). In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 23–33. Springer, Heidelberg (1988)
17. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM Review 45(4), 727–784 (2003)
18. Digital signature standard (DSS). Federal Information Processing Standard, Publication 186-2, U.S. Department of Commerce, National Institute of Standards and Technology (2000)
19. Digital signature standard (DSS). Federal Information Processing Standard, Publication 186, U.S. Department of Commerce, National Institute of Standards and Technology (1994)
20. ANSI X9.62. Public Key Cryptography for the Financial Service Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). American National Standard Institute. American Bankers Associtaion (1998)
21. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
22. 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)
23. Gennaro, R., Krawczyk, H., Rabin, T.: RSA-Based Undeniable Signatures. Journal of Cryptology 13(4), 397–416 (2000)
24. Goldreich, O., Kahan, A.: How To Construct Constant-Round Zero-Knowledge Proof Systems for NP. Journal of Cryptology 9(3), 167–189 (1996)
25. Goldreich, O., Micali, S., Wigderson, A.: Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. Journal of the ACM 38(1), 691–729 (1991)
26. Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof-Systems. In: STOC 1985, pp. 291–304. ACM Press, New York (1985)
27. Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof Systems. SIAM Journal on Computing 18(1), 186–208 (1989)
28. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing 17(2), 281–308 (1988)
29. Guillou, L.C., Quisquater, J.-J.: A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Trasmission and Memory. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 123–128. Springer, Heidelberg (1988)
30. 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)

31. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated Verifier Proofs and Their Applications. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 143–154. Springer, Heidelberg (1996)
32. Li, J., Wang, Y.: Universal Designated Verifier Ring Signature (Proof) Without Random Oracles. In: Zhou, X., Sokolsky, O., Yan, L., Jung, E.-S., Shao, Z., Mu, Y., Lee, D.C., Kim, D.Y., Jeong, Y.-S., Xu, C.-Z. (eds.) EUC Workshops 2006. LNCS, vol. 4097, pp. 332–341. Springer, Heidelberg (2006)
33. Monnerat, J., Vaudenay, S.: Generic Homomorphic Undeniable Signatures. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 354–371. Springer, Heidelberg (2004)
34. Monnerat, J., Vaudenay, S., Vuagnoux, M.: About Machine-Readable Travel Documents – Privacy Enhancement Using (Weakly) Non-Transferable Data Authentication. In: RFIDSEC 2007 (2007)
35. Machine Readable Travel Documents. Development of a Logical Data Structure — LDS For Optional Capacity Expansion Technologies. Version 1.7 (2004), http://www.icao.int/mrtd/download/technical.cfm
36. Machine Readable Travel Documents. PKI for Machine Readable Travel Documents offering ICC Read-Only Access. Version 1.1 (2004), http://www.icao.int/mrtd/download/technical.cfm
37. Ogata, W., Kurosawa, K., Heng, S.-H.: The Security of the FDH Variant of Chaum's Undeniable Signature Scheme. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 328–345. Springer, Heidelberg (2005)
38. Okamoto, T., Ohta, K.: How to Utilize the Randomness of Zero-Knowledge Proofs. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 456–475. Springer, Heidelberg (1991)
39. 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)
40. Pass, R.: On Deniability in the Common Reference String and Random Oracle Model. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 316–337. Springer, Heidelberg (2003)
41. Pass, R.: Alternative Variants of Zero-Knowledge Proofs. Licentiate Thesis (2004)
42. Schnorr, C.-P.: Efficient Identification and Signatures for Smart Cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (1990)
43. Schnorr, C.-P.: Efficient Signature Generation by Smart Cards. Journal of Cryptology 4(3), 161–174 (1991)
44. Shahandashti, S.F., Safavi-Naini, R.: Construction of Universal Designated-Verifier Signatures and Identity-Based Signatures from Standard Signatures. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 121–140. Springer, Heidelberg (2008)
45. Shahandashti, S.F., Safavi-Naini, R., Baek, J.: Concurrently-secure credential ownership proofs. In: ASIACCS 2007, pp. 161–172. ACM Press, New York (2007)
46. Steinfeld, R., Bull, L., Wang, H., Pieprzyk, J.: Universal Designated-Verifier Signatures. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 523–542. Springer, Heidelberg (2003)
47. Vaudenay, S.: E-Passport Threats. IEEE Security and Privacy Magazine 5(6), 61–64 (2007)
48. Vaudenay, S., Vuagnoux, M.: About Machine-Readable Travel Documents. In: ICS 2007. LNCS. Springer, Heidelberg (2007)

Homomorphic MACs: MAC-Based Integrity for Network Coding

Shweta Agrawal[1,⋆] and Dan Boneh[2,⋆⋆]

[1] The University of Texas at Austin
shweta.a@gmail.com
[2] Stanford University
dabo@cs.stanford.edu

Abstract. Network coding has been shown to improve the capacity and robustness in networks. However, since intermediate nodes modify packets en-route, integrity of data cannot be checked using traditional MACs and checksums. In addition, network coded systems are vulnerable to pollution attacks where a single malicious node can flood the network with bad packets and prevent the receiver from decoding the packets correctly. Signature schemes have been proposed to thwart such attacks, but they tend to be too slow for online per-packet integrity.

Here we propose a *homomorphic MAC* which allows checking the integrity of network coded data. Our homomorphic MAC is designed as a drop-in replacement for traditional MACs (such as HMAC) in systems using network coding.

1 Introduction

Network coding [1,2] proposes to replace the traditional 'store and forward' paradigm in networks by more intelligent routing that allows intermediate nodes to transform the data in transit. Network coding has become popular due to its robustness and the improved throughput it offers.

When transmitting a message using linear network coding [3] the sender first breaks the message into a sequence of m vectors $\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_m$ in an n-dimensional linear space $\mathbb{F}_q^n$, where n, m and q are fixed ahead of time. Often $q = 2^8$ so that the entire transmitted message is $n \times m$ bytes. The sender transmits these message vectors to its neighboring nodes in the network. As the vectors traverse the network, moving from one node to the next on their way to the destination, the nodes randomly combine the vectors with each other. More precisely, each node in the network creates a random linear combination of the vectors it receives and transmits the resulting linear combination to its adjacent nodes. Intended recipients thus receive random linear combinations of the original message vectors. Recipients can recover the original message from any set of m random linear combinations that form a full rank matrix.

⋆ Supported by DARPA IAMANET.

⋆⋆ Supported by DARPA IAMANET, NSF, and the Packard Foundation.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 292–305, 2009.

For this approach to work, every vector $\hat{\mathbf{v}}$ in the network must carry with it the coefficients $\alpha_1, \ldots, \alpha_m \in \mathbb{F}_q$ that produce $\hat{\mathbf{v}}$ as a linear combination of the original message vectors. To do so, prior to transmission, the source node augments every message vector $\hat{\mathbf{v}}_i$ with m additional components. The resulting vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$, called *augmented vectors*, are given by:

$$\mathbf{v}_i = (\text{—}\hat{\mathbf{v}}_i\text{—},\ \overbrace{\underbrace{0, \ldots, 0, 1}_{i}, 0, \ldots, 0}^{m}\ ) \quad \in \mathbb{F}_q^{n+m} \tag{1}$$

i.e., each original vector $\hat{\mathbf{v}}_i$ is appended with the vector of length m containing a single '1' in the ith position. These augmented vectors are then sent by the source as packets in the network. Observe that if $\mathbf{y} \in \mathbb{F}_q^{n+m}$ is a linear combination of $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{F}_q^{n+m}$ then the linear combination coefficients are contained in the last m coordinates of $\mathbf{y}$.

Pollution Attacks. Our brief description of linear network coding assumes all nodes are honest. However, if some nodes are malicious and forward invalid linear combinations of received vectors, then recipients obtain multiple packets, only some of which are proper linear combinations of the original message vectors. In such a scenario, recipients have no way of telling which of their received vectors are corrupt and should be ignored during decoding.

Detailed discussion of pollution attacks can be found in [4,5,6]. Here we only note that pollution attacks cannot be mitigated by standard signatures or MACs. Clearly, signing the augmented message vectors is of no use since recipients do not have the original message vectors and therefore cannot verify the signature. Similarly, signing the entire message prior to transmission does not work. To see why, observe that recipients can obtain multiple vectors where, say, only half are proper linear combinations of the original message vectors and the other half are corrupt. Recipients would need to decode exponentially many m-subsets until they find a decoded message that is consistent with the signature (decoding produces the correct transmitted message only when all m vectors being decoded are a linear combination of the original message vectors). In summary, as explained in [4,5,6], new integrity mechanisms are needed to mitigate pollution attacks.

Previous Solutions. Recently, several approaches have been proposed to thwart pollution attacks. Of these, some solutions are information theoretic while others are cryptographic. We refer to [4] for a survey of defenses. Here we restrict our attention to cryptographic solutions. Several authors [4,5,7,8] devised digital signature schemes for signing a linear subspace. Let V be the linear space spanned by the augmented message vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{F}_q^{n+m}$ that the sender transmits. These signature schemes produce a signature σ on V such that $\mathsf{Verify}(PK, \mathbf{v}, \sigma)$ holds for every $\mathbf{v} \in V$, but it is difficult to construct a vector $\mathbf{y} \notin V$ for which $\mathsf{Verify}(PK, \mathbf{y}, \sigma)$ holds. Recipients use these signatures to reject all received vectors that are not in the subspace V, mitigating the pollution problem.

The digital signature constructions in [4,5,7,8] are very elegant and very appropriate for offline network coding systems, such as robust distributed file storage [9]. For online traffic, however, these systems are too slow to sign every packet. A different solution is needed if one is to defend against pollution attacks at line speeds.

Another difficulty with existing digital signatures is that they require the network coding coefficients to live in a field $\mathbb{F}_q$ where q is the order of a group where discrete-log is difficult, e.g. $q \approx 2^{160}$. Therefore transmitting each coefficient requires 20 bytes and hence the augmentation components add $20 \times m$ bytes to every packet. Recall that in typical linear network coding (without integrity) $q = 2^8$ so that the augmentation components add only m bytes to every packet. Again, a different solution is needed if one wishes to minimize the augmentation overhead.

Faster methods for network coding integrity were studied in [10,11]. In [10], the authors extend the idea of using homomorphic hashes, first proposed in [8]. They suggest clever batch mechanisms and co-operation of well-behaved users in order to reduce the cost of online verification. This method, however, requires a client to separately download hashes of every file that it wishes to download. The lightweight homomorphic hashing scheme of [11] also entails the overhead of downloading file hashes separately as in [10]. In addition, it requires new hashes to be computed for every client that joins the system, and assumes a secure channel between the server and client for communicating these hashes.

Our Contribution. We design a MAC scheme that can be used to mitigate pollution attacks. We construct the MAC in three steps.

First, we construct a homomorphic MAC. That is, given two (vector,tag) pairs $(\mathbf{v}_1, t_1)$ and $(\mathbf{v}_2, t_2)$, where $v_1, v_2 \in \mathbb{F}_q^{n+m}$, anyone can create a valid tag t for the vector $\mathbf{y} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2$ for any $\alpha_1, \alpha_2 \in \mathbb{F}_q$. Roughly speaking, our security proof shows that, even under a chosen message attack, creating a valid tag for a vector outside the linear span of the original message vectors is difficult. We give the details in Section 2 and 3. Our MAC is related to the classic MAC of Carter and Wagman [12].

This MAC system can be used to mitigate pollution attacks when the source and recipient have a shared secret key. The source uses the secret key to compute a tag for each of the original message vectors. Intermediate nodes then use the homomorphic property to compute valid tags for random linear combinations they produce. The recipient verifies the tags of received vectors and drops all vectors with an invalid tag.

Network coding (for a single source) is most useful in broadcast settings where there are multiple recipients for every message. Using our basic homomorphic MAC the sender would need to have a shared secret key with each recipient. In addition, if we want intermediate nodes to verify tags before forwarding them on to other nodes, then the sender would need to have a shared secret key with each network node. Every packet would need to include a tag per network node, which is unworkable.

Our second step converts the homomorphic MAC into a *broadcast* homomorphic MAC using a technique of Canetti et al. [13]. This enables any network node to validate vectors it receives. The limitation of this construction is that it is only c-collusion resistant for some pre-determined c. That is, when more than

c recipients and intermediate nodes collude, the MAC becomes insecure. In some settings, it may be possible to apply TESLA-like methods [14] to convert our homomorphic MAC into a broadcast MAC. We do not explore this here since we want intermediate network nodes to verify packet integrity *before* forwarding packets.

Our third step converts the broadcast homomorphic MAC into an integrity system where there are multiple senders and multiple verifiers. The result is a network coding MAC in networks where anyone can be either a sender, a recipient, or an intermediate node.

We experimented with our construction and give running times in Section 7.

Notation: Throughout the paper we let $[m]$ denote the set of integers $\{1, \ldots, m\}$. For vectors $\mathbf{u}$ and $\mathbf{v}$ in $\mathbb{F}_q^n$ we let $\mathbf{u} \cdot \mathbf{v} \in \mathbb{F}_q$ denote the inner product of $\mathbf{u}$ and $\mathbf{v}$.

2 Homomorphic MACs: Definitions

We begin by defining homomorphic MACs and their security. A (q, n, m) *homomorphic MAC* is defined by three probabilistic, polynomial-time algorithms, (Sign, Verify, $\mathsf{Combine}$). The Sign algorithm computes a tag for a vector space $V = \text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_m)$ by computing a tag for one basis vector at a time. $\mathsf{Combine}$ implements the homomorphic property and Verify verifies vector-tag pairs. Each vector space V is identified by an identifier id which is chosen arbitrarily from a set $\mathcal{I}$. In more detail, these algorithms provide the following functionality:

- $\mathsf{Sign}(k, \mathsf{id}, \mathbf{v}, i)$: Input: a secret key k, a vector space identifier id, an augmented vector $\mathbf{v} \in \mathbb{F}_q^{n+m}$, and $i \in [m]$ indicating that $\mathbf{v}$ is the ith basis vector of the vector space identified by id.
  Output: tag t for $\mathbf{v}$.

  As explained above, the Sign algorithm signs a vector space $V \subseteq \mathbb{F}_q^{n+m}$ spanned by $\mathbf{v}_1, \ldots, \mathbf{v}_m$ by running $\mathsf{Sign}(k, \mathsf{id}, \mathbf{v}_i, i)$ for $i = 1, \ldots, m$. This produces a tag for each of the basis vectors. The identifier id identifies the vector space V. When transmitting a vector $\mathbf{v}_i$ into the network, the sender transmits $(\mathsf{id}, \mathbf{v}_i, t_i)$. Recipients collect all valid vectors with a given id and decode those as a group to obtain the original basis vectors encoding the transmitted message. We let $\mathcal{I}$ denote the set of identifiers and $\mathcal{K}$ denote the set of keys.

- $\mathsf{Combine}((\mathbf{v}_1, t_1, \alpha_1), \ldots, (\mathbf{v}_m, t_m, \alpha_m))$: Input: m vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{F}_q^{n+m}$ and their tags $t_1, \ldots, t_m$ under key k plus m constants $\alpha_1, \ldots, \alpha_m \in \mathbb{F}_q$.
  Output: a tag t on the vector $\mathbf{y} := \sum_{i=1}^m \alpha_i \mathbf{v}_i \ \in \mathbb{F}_q^{n+m}$.

- $\mathsf{Verify}(k, \mathsf{id}, \mathbf{y}, t)$: Input: a secret key k, an identifier id, a vector $\mathbf{y} \in \mathbb{F}_q^{n+m}$, and a tag t.
  Output: 0 (reject) or 1 (accept).

We require that the scheme satisfy the following correctness property. Let V be an m-dimensional subspace of $\mathbb{F}_q^{n+m}$ with basis $\mathbf{v}_1, \ldots, \mathbf{v}_m$ and identifier id. Let $k \in \mathcal{K}$ and $t_i := \mathsf{Sign}(k, \mathsf{id}, \mathbf{v}_i, i)$ for $i = 1, \ldots, m$. Let $\alpha_1, \ldots, \alpha_m \in \mathbb{F}_q$. Then

$$\mathsf{Verify}\left(k,\ \mathsf{id},\ \sum_{i=1}^{m}\alpha_i\mathbf{v}_i,\ \mathsf{Combine}\big((\mathbf{v}_1,t_1,\alpha_1),\ldots,(\mathbf{v}_m,t_m,\alpha_m)\big)\right)=1$$

Security. Next, we define security for homomorphic MACs. We allow the attacker to obtain the signature on arbitrary vector spaces of its choice (analogous to a chosen message attack on MACs). Each vector space V_i submitted by the attacker has an identifier id_i. The attacker should be unable to produce a valid triple $(\mathsf{id},\mathbf{y},t)$ where either id is new or $\mathsf{id}=\mathsf{id}_i$ but $\mathbf{y}\notin V_i$. More precisely, we define security using the following game (a similar game is used to define secure homomorphic signatures in [4]):

Attack Game 1. Let $\mathcal{T}=(\mathsf{Sign},\mathsf{Combine},\mathsf{Verify})$ be a (q,n,m) homomorphic MAC. We define security of $\mathcal{T}$ using the following game between a challenger C and an adversary $\mathcal{A}$.

Setup. The challenger generates a random key $k\overset{\mathrm{R}}{\leftarrow}\mathcal{K}$.

Queries. $\mathcal{A}$ adaptively submits MAC queries where each query is of the form (V_i,id_i) where V_i is a linear subspace (represented by a basis of m vectors) and id_i is a space identifier. We require that all identifiers id_i submitted by $\mathcal{A}$ are distinct. To respond to a query for (V_i,id_i) the challenger does:

```
Let $\mathbf{v}_1,\ldots,\mathbf{v}_m\in\mathbb{F}_q^{n+m}$ be a basis for $V_i$
//   Now compute MAC for all basis vectors:
for $j=1,\ldots,m$ let $t_j\overset{\mathrm{R}}{\leftarrow}\mathsf{Sign}(k,\mathsf{id}_i,\mathbf{v}_j,j)$
send $(t_1,\ldots,t_m)$ to $\mathcal{A}$
```

Output. The adversary $\mathcal{A}$ outputs an identifier id^*, a tag t^*, and a vector $\mathbf{y}^*\in\mathbb{F}_p^{n+m}$.

The adversary *wins* the security game if $\mathsf{Verify}(k,\mathsf{id}^*,\mathbf{y}^*,t^*)=1$, and either

1. $\mathsf{id}^*\neq\mathsf{id}_i$ for all i (a *type 1 forgery*), or
2. $\mathsf{id}^*=\mathsf{id}_i$ for some i and $\mathbf{y}^*\notin V_i$ (a *type 2 forgery*)

Moreover, let $\mathbf{y}^*=(y_1^*,\ldots,y_{n+m}^*)$. Then the augmentation $(y_{n+1}^*,\ldots,y_{n+m}^*)$ in $\mathbf{y}^*$ is not the all zero vector (which corresponds to a trivial forgery).

The *advantage* NC-Adv$[\mathcal{A},\mathcal{T}]$ of $\mathcal{A}$ with respect to $\mathcal{T}$ is defined to be the probability that $\mathcal{A}$ wins the security game.

Definition 1. A (q,n,m) homomorphic MAC scheme $\mathcal{T}$ is secure if for all polynomial time adversaries $\mathcal{A}$ the quantity NC-Adv$[\mathcal{A},\mathcal{T}]$ is negligible.

3 Construction 1: A Homomorphic MAC

In this section we describe a secure homomorphic MAC. Our construction is derived from a classic MAC system due to Carter and Wagman [12]. Shacham and Waters [15] recently proposed another homomorphic MAC, also based on the Carter-Wagman MAC, but the motivation, setup and security game are very different from ours.

The MAC Scheme HomMac*:* To construct a (q, n, m) homomorphic MAC we use a Pseudo Random Generator $G : \mathcal{K}_{\mathrm{G}} \to \mathbb{F}_q^{n+m}$ and a Pseudo Random Function $F : \mathcal{K}_{\mathrm{F}} \times (\mathcal{I} \times [m]) \to \mathbb{F}_q$. Keys for our MAC consist of pairs (k_1, k_2) where $k_1 \in \mathcal{K}_{\mathrm{G}}$ and $k_2 \in \mathcal{K}_{\mathrm{F}}$. The MAC works as follows:

- $\mathsf{Sign}(k, \mathsf{id}, \mathbf{v}, i)$: To generate a tag for an ith basis vector $\mathbf{v} \in \mathbb{F}_q^{n+m}$ using key $k = (k_1, k_2)$ do:
  (1) $\mathbf{u} \leftarrow G(k_1) \ \in \mathbb{F}_q^{n+m}$
  (2) $b \leftarrow F\big(k_2,\ (\mathsf{id}, i)\big) \in \mathbb{F}_q$
  (3) $t \leftarrow (\mathbf{u} \cdot \mathbf{v}) + b \quad \in \mathbb{F}_q$
  Output t. Note that the tag is a single element of $\mathbb{F}_q$.
- $\mathsf{Combine}((\mathbf{v}_1, t_1, \alpha_1), \ldots, (\mathbf{v}_m, t_m, \alpha_m))$: output $t \leftarrow \sum_{j=1}^{m} \alpha_j t_j \ \in \mathbb{F}_q$.
- $\mathsf{Verify}(k, \mathsf{id}, \mathbf{y}, t)$: Let $k = (k_1, k_2)$ be a secret key and let $\mathbf{y} = (y_1, \ldots, y_{n+m}) \in \mathbb{F}_q^{n+m}$.
  Do the following:
  $\mathbf{u} \leftarrow G(k_1) \ \in \mathbb{F}_q^{n+m} \qquad \text{and} \qquad a \leftarrow (\mathbf{u} \cdot \mathbf{y}) \in \mathbb{F}_q$
  $b \leftarrow \sum_{i=1}^{m} \big[y_{n+i} \cdot F\big(k_2,\ (\mathsf{id}, i)\big)\big] \in \mathbb{F}_q$
  if $a + b = t$ output 1; otherwise output 0

This completes the description of HomMac. To verify correctness of the scheme, suppose $\mathbf{y} = \sum_{i=1}^{m} \alpha_i \mathbf{v}_i$ where $\mathbf{v}_1, \ldots, \mathbf{v}_m$ are the original augmented basis vectors and $t_1, \ldots, t_m$ are their tags. The coordinates $(y_{n+1}, \ldots, y_{n+m})$ of $\mathbf{y}$ are equal to the coefficients $(\alpha_1, \ldots, \alpha_m)$. Therefore, $a + b$ computed in Verify satisfies

$$a + b = \mathbf{u} \cdot \mathbf{y} + b = \sum_{i=1}^{m} \alpha_i \cdot \big(\ (\mathbf{u} \cdot \mathbf{v}_i) + F(k_2,\ (\mathsf{id}, i))\ \big) = \sum_{i=1}^{m} \alpha_i \cdot t_i$$

which is precisely the output of $\mathsf{Combine}\big((\mathbf{v}_1, t_1, \alpha_1), \ldots, (\mathbf{v}_m, t_m, \alpha_m)\big)$, as required.

Security. We prove security assuming G is a secure PRG and F is a secure PRF. For a PRF adversary $\mathcal{B}_1$ we let PRF-Adv$[\mathcal{B}_1, F]$ denote $\mathcal{B}_1$'s advantage in winning the PRF security game with respect to F. Similarly, for a PRG adversary $\mathcal{B}_2$ we let PRG-Adv$[\mathcal{B}_2, G]$ be $\mathcal{B}_2$'s advantage in winning the PRG security game with respect to G. We refer to [16] for a definition of the PRF and PRG security games.

Theorem 2. *For any fixed q, n, m, the MAC scheme* HomMac *is a secure (q, n, m) homomorphic MAC assuming the PRG G is a secure PRG and the PRF F is a secure PRF.*

In particular, for all homomorphic MAC adversaries $\mathcal{A}$ there is a PRF adversary $\mathcal{B}_1$ and a PRG adversary $\mathcal{B}_2$ (whose running times are about the same as that of $\mathcal{A}$) such that

$$\text{NC-Adv}[\mathcal{A}, \mathit{HomMac}] \leq \text{PRF-Adv}[\mathcal{B}_1, F] + \text{PRG-Adv}[\mathcal{B}_2, G] + (1/q)$$

Proof. We prove the theorem using a sequence of three games denoted Game 0,1,2. For $i = 0, 1, 2$ let W_i be the event that $\mathcal{A}$ wins the homomorphic MAC security game in Game i.

Game 0 is identical to Attack Game 1 applied to the scheme HomMac. Therefore

$$\Pr[W_0] = \text{NC-Adv}[\mathcal{A}, \mathsf{HomMac}] \tag{2}$$

In Game 1 we replace the output of the PRG used in HomMac with a truly random string. That is, Game 1 is identical to Game 0 except that to respond to MAC queries the challenger computes at initialization time $\mathbf{u} \overset{\text{R}}{\leftarrow} \mathbb{F}_q^{n+m}$ instead of $\mathbf{u} \leftarrow G(k_1)$ in step (1) of the Sign algorithm. Everything else remains the same. Then there is a PRG adversary $\mathcal{B}_2$ such that

$$\big|\Pr[W_0] - \Pr[W_1]\big| = \text{PRG-Adv}[\mathcal{B}_2, G] \tag{3}$$

In Game 2 we replace the PRF by a truly random function. That is, Game 2 is identical to Game 1 except that to respond to MAC queries the challenger computes $b \overset{\text{R}}{\leftarrow} \mathbb{F}_q$ instead of $b \leftarrow F\big(k_2,\ (\mathsf{id}_i, j)\big)$ in step (2) of the Sign algorithm. Everything else remains the same. Then there is a PRF adversary $\mathcal{B}_1$ such that

$$\big|\Pr[W_1] - \Pr[W_2]\big| = \text{PRF-Adv}[\mathcal{B}_1, F] \tag{4}$$

The complete challenger in Game 2 works as follows:

Initialization: $\mathbf{u} \overset{\text{R}}{\leftarrow} \mathbb{F}_q^{n+m}$

The adversary submits MAC queries (V_i, id_i) where $V_i = \text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_m)$ is a subspace of $\mathbb{F}_q^{n+m}$.

The challenger responds to query number i as follows:

```
for $j = 1, \ldots, m$ do:
    $b_{i,j} \overset{\text{R}}{\leftarrow} \mathbb{F}_q$    and    $t_{i,j} \leftarrow (\mathbf{u} \cdot \mathbf{v}_j) + b_{i,j} \quad \in \mathbb{F}_q$
send $(t_{i,1}, \ldots, t_{i,m})$ to $\mathcal{A}$
```

Eventually the adversary outputs $(\mathsf{id}^*, t^*, \mathbf{y}^*)$. To determine if the adversary wins the game we first compute:

```
if $\mathsf{id}^* = \mathsf{id}_i$ then
set   $(b_1^*, \ldots, b_m^*) \leftarrow (b_{i,1}, \ldots, b_{i,m})$        //   (type 2 forgery)
else for $j = 1, \ldots, m$ set   $b_j^* \overset{\text{R}}{\leftarrow} \mathbb{F}_q$        //   (type 1 forgery)
```

Let $\mathbf{y}^* = (y_1^*, \ldots, y_{n+m}^*)$. The adversary wins (i.e. event W_2 happens) if

$$t^* = (\mathbf{u} \cdot \mathbf{y}^*) + \sum_{j=1}^{m} (y_{n+j}^* \cdot b_j^*) \tag{5}$$

and, for a type 2 forgery $\mathbf{y}^* \notin V_i$. Moreover the augmentation $(y_{n+1}^*, \ldots, y_{n+m}^*)$ in $\mathbf{y}^*$ is not all zero.

We now show that $\Pr[W_2] = 1/q$ in Game 2. This is the crux of the proof. Let T be the event that the adversary outputs a type 1 forgery.

Type 1 forgery (event T happens): We bound $\Pr[W_2 \wedge T]$. In a type 1 forgery the right hand side of (5) is a random value in $\mathbb{F}_q$ independent of the adversary's view. Therefore, when event T happens, the probability that (5) holds is exactly $1/q$. Hence, $\Pr[W_2 \wedge T] = (1/q) \cdot \Pr[T]$.

Type 2 forgery (event $\neg T$ happens): We bound $\Pr[W_2 \wedge \neg T]$. In a type 2 forgery $\mathcal{A}$ uses an id^* used in one of the MAC queries. Then $\mathsf{id}^* = \mathsf{id}_i$ for some i. Event W_2 happens if $\mathbf{y}^* \notin V_i$ and (5) holds.

Let $\{t'_1, \dots, t'_m\}$ be the tags for the basis vectors $\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$ of the linear space V_i. Define

$$\mathbf{y}' := \sum_{j=1}^{m} y^*_{n+j} \cdot \mathbf{v}_j \in V_i \quad \text{and} \quad t' := \sum_{j=1}^{m} y^*_{n+j} \cdot t'_j \in \mathbb{F}_q$$

Then, t' is a valid tag for $\mathbf{y}'$. Hence, we now know that the following two relations hold:

$$(\mathbf{u} \cdot \mathbf{y}^*) + \sum_{j=1}^{m} y^*_{n+j} \cdot b_{i,j} = t \tag{6}$$

$$(\mathbf{u} \cdot \mathbf{y}') + \sum_{j=1}^{m} y'_{n+j} \cdot b_{i,j} = t' \tag{7}$$

Recall that $\mathbf{v}_1, \dots, \mathbf{v}_m$ are properly augmented vectors and therefore $y'_{n+j} = y^*_{n+j}$ for $j = 1, \dots, m$. Subtracting (7) from (6) we obtain

$$\big(\mathbf{u} \cdot (\mathbf{y}^* - \mathbf{y}')\big) = t - t' \tag{8}$$

Hence, by producing a valid forgery, the adversary found a $\mathbf{y}^*$ and t that satisfy (8). Moreover, since $\mathbf{y}^* \notin V_i$ but $\mathbf{y}' \in V_i$ we know that $\mathbf{y}^* \neq \mathbf{y}'$. But since in the adversary's view, $\mathbf{u}$ is indistinguishable from a random vector in $\mathbb{F}_q^{n+m}$, the probability that he can satisfy (8) is exactly $1/q$. Hence, when event $\neg T$ happens one can show that $\Pr[W_2 \wedge \neg T] = (1/q) \cdot \Pr[\neg T]$.

Putting together our bounds for $\Pr[W_2 \wedge T]$ and $\Pr[W_2 \wedge \neg T]$ we obtain

$$\Pr[W_2] = \Pr[W_2 \wedge T] + \Pr[W_2 \wedge \neg T] = 1/q(\Pr[T] + \Pr[\neg T]) = 1/q \tag{9}$$

Putting together equations (2),(3),(4),(9) proves the theorem. □

Improved Security. Since the tag on a vector $\mathbf{v}$ is a single element in $\mathbb{F}_q$, there is a homomorphic MAC adversary that can break the MAC (i.e. win the MAC security game) with probability $1/q$. When $q = 2^8$ the MAC can be broken with probability $1/256$. Security can be improved by computing multiple MACs per data vector. For example, with 8 tags per vector security becomes $1/q^8$. For

$q = 2^8$ the resulting tag is 8 bytes long. The proof of Theorem 2 easily extends to prove these bounds for HomMac using multiple tags.

We note, however, that a homomorphic MAC with security 1/256 may be sufficient for the network coding application. The reason is that the homomorphic MAC is only used by recipients to drop malformed received vectors. The sender can, in addition, compute a regular MAC (such as HMAC) on the transmitted message prior to encoding it using network coding. Recipients, after decoding a matrix of vectors with valid homomorphic MACs, will further validate the HMAC on the decoded message and drop the message if its HMAC is invalid. Hence, success in defeating the homomorphic MAC does not mean that a rogue message is accepted by recipients. It only means that recipients may need to do a little more work to properly decode the message (by trying various m-subsets of the received vectors with a valid homomorphic MAC). As mentioned above, this issue can be avoided by increasing the security of the homomorphic MAC by computing multiple tags per vector.

4 Broadcast Homomorphic MACs: Definitions

We next convert the homomorphic MAC of the previous section to a broadcast homomorphic MAC. This will enable all nodes in the network (both recipients and routers) to verify tags in transmitted packets. We start by defining security for a broadcast homomorphic tag, which takes into account a set of nodes trying to fool some other node.

A broadcast homomorphic MAC is 0 by a five tuple (q, n, m, μ, c) where (q, n, m) are as in the previous section, μ is the number of nodes in the system, and c is the collusion bound (the maximum number of nodes that can collude to fool another node).

A (q, n, m, μ, c) *broadcast homomorphic MAC* is defined by four probabilistic, polynomial-time algorithms, (Setup, Sign, Verify, Combine) that provide the following functionality:

- Setup(λ, μ, c): Input: security parameter λ, number of users in the system μ, and desirable collusion resistance bound c. Output: A set of $\mu + 1$ keys $k, k_1, \ldots, k_\mu$. Here k is the sender's key and $k_1, \ldots, k_\mu$ are keys given to the μ verifiers.
- Algorithms Sign, Combine, Verify are as in Section 2, except that the Sign algorithm is given the key k and the Verify algorithm is given one of the keys k_i for some $i \in [\mu]$.

The system must satisfy a correctness requirement analogous to the one in Section 2.

Security: Next, we define security against c-collusions. The adversary $\mathcal{A}$ is given c verifier keys and its goal is to create a message-tag pair that will verify under some verifier's key not in the adversary's possession. More precisely, we define security using the following game.

Attack Game 2. Let $\mathcal{T} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Verify})$ be a (q, n, m, μ, c) broadcast homomorphic MAC. We define security of $\mathcal{T}$ using the following game between a challenger C and an adversary A (the security parameter λ is given as input to both the challenger and the adversary).

Setup. The adversary sends the challenger the indices of c users acting as verifiers $\{i_1, \ldots, i_c\}$. The challenger runs $\mathsf{Setup}(\lambda, c, \mu)$ to obtain keys $k, k_1, \ldots, k_\mu$ and sends the keys $\{k_{i_1}, \ldots, k_{i_c}\}$ to the adversary.

Queries. The adversary adaptively submits MAC queries as in Attack Game 1. The challenger responds as in that game using the sender's key k.

Output. The adversary $\mathcal{A}$ outputs an index $j^* \in [\mu] \setminus \{i_1, \ldots, i_c\}$, an identifier id^*, a tag t^*, and a vector $\mathbf{y}^* \in \mathbb{F}_p^{n+m}$.

The adversary *wins* the security game if $\mathsf{Verify}(k_{j^*}, \mathsf{id}^*, \mathbf{y}^*, t^*) = 1$, and the additional winning conditions of Attack Game 1 are satisfied.

The *advantage* BNC-Adv$[\mathcal{A}, \mathcal{T}]$ of $\mathcal{A}$ with respect to $\mathcal{T}$ is defined to be the probability that $\mathcal{A}$ wins this security game.

Definition 3. A (q, n, m, μ, c) broadcast homomorphic MAC scheme $\mathcal{T}$ is secure if for all polynomial time adversaries $\mathcal{A}$, the quantity BNC-Adv$[\mathcal{A}, \mathcal{T}]$ is negligible.

5 Construction 2: A Broadcast Homomorphic MAC

We convert our homomorphic MAC HomMac into a broadcast MAC using a technique of Canetti et al. [13] based on cover free set systems. Instead of computing one tag per vector, we compute several tags per vector using independent keys. We give each verifier a subset of all MAC keys. Thus, each verifier can validate a subset of the MACs on each packet. More importantly, when key assignment is done properly, no coalition of c verifiers can fool another verifier. We start by recalling a few definitions.

Definition 4. A *set system* is a pair $(\mathbb{X}, \mathbb{B})$ where $\mathbb{X}$ is a finite set of elements and $\mathbb{B} = (A_1, \ldots, A_\mu)$ is an ordered set of subsets of $\mathbb{X}$.

Definition 5. A set system $(\mathbb{X}, \mathbb{B})$ is called a (c, d)*−cover free family* if for all c distinct sets $A_1, \ldots, A_c \in \mathbb{B}$ and any other set $A \in \mathbb{B}$, we have $|A \setminus \cup_{j=1}^{c} A_j| > d$.

We construct a (q, n, m, μ, c) broadcast homomorphic MAC from HomMac and any (c, d) cover free family $(\mathbb{X}, \mathbb{B})$ where $|\mathbb{B}| = \mu$. The parameter d is important for security; the error term in the security proof is $(1/q)^d$. The system works as follows:

MAC Scheme BrdctHomMac:

$\mathsf{Setup}(\lambda, c, \mu)$: Pick a (c, d) cover free family $(\mathbb{X}, \mathbb{B})$, such that $|\mathbb{B}| = \mu$ and $\frac{1}{q^d} < \frac{1}{2^\lambda}$.

Let $\ell = |\mathbb{X}|$ and generate ℓ keys $\{K_1, \dots, K_\ell\}$ for HomMac. We equate $\mathbb{X}$ with this set of keys, i.e. $\mathbb{X} := \{K_1, \dots, K_\ell\}$.

The sender's key k consists of all ℓ keys in $\mathbb{X}$. We assign to verifier number i (where $i \in [\mu]$) the key $k_i := A_i \subseteq \mathbb{X}$ where A_i is subset number i in $\mathbb{B}$.

$\mathsf{Sign}(\mathbb{X}, \mathsf{id}, \mathbf{v}, i)$: For $j = 1, \dots, \ell$ compute $t_j \leftarrow \mathsf{HomMac\text{-}Sign}(K_j, \mathsf{id}, \mathbf{v}, i)$ and output $t := (t_1, \dots, t_\ell)$.

$\mathsf{Combine}((\mathbf{v}_1, t_1, \alpha_1), \dots, (\mathbf{v}_m, t_m, \alpha_m))$: Apply $\mathsf{HomMac\text{-}Combine}$ to all ℓ tags in the m tuples.

$\mathsf{Verify}(A_i, \mathsf{id}, \mathbf{y}, t)$: Here t is a tuple of ℓ tags. The key A_i is a set of b keys for HomMac, say $\{K_{i,1}, \dots, K_{i,b}\}$.

```
Do the following:
    Select the $b$ tags in the tuple $t$ that correspond to the $b$ keys in $A_i$.
    Call them $t_1, \dots, t_b$.
    For $j = 1, \dots, b$,
    $r_j \leftarrow \mathsf{HomMac\text{-}Verify}(K_{i,j}, \mathsf{id}, \mathbf{y}, t_j)$
    If $r_j = 1$ for all $j \in [b]$, output 1; otherwise output 0
```

Security. The following simple theorem states the security property of this construction. Recall that HomMac uses a PRF and a PRG.

Theorem 6. *For any fixed q, n, m, μ, c, the broadcast homomorphic MAC $\mathit{BrdctHomMac}$ is a secure (q, n, m, μ, c) Broadcast Homomorphic MAC assuming the PRG G is a secure PRG and the PRF F is a secure PRF.*

In particular, for all broadcast homomorphic MAC adversaries $\mathcal{A}$ there is a PRF adversary $\mathcal{B}_1$ and a PRG adversary $\mathcal{B}_2$ (whose running times are about the same as that of $\mathcal{A}$) such that

$$\text{BNC-Adv}[\mathcal{A}, \mathit{BrdctHomMac}] \le \text{PRF-Adv}[\mathcal{B}_1, F] + \text{PRG-Adv}[\mathcal{B}_2, G] + (1/q)^d \tag{10}$$

The proof is straight forward from Theorem 2 and is omitted here. Since q is fairly small (e.g. $q = 256$) it is very important that the error term in (10) is $(1/q)^d$. In particular, one needs a (c, d) cover free set system where d makes $(1/q)^d$ negligible (or concretely $(1/q)^d < (1/2)^\lambda$). The $(1/q)^d$ error term is obtained thanks to properties of the HomMac homomorphic MAC.

6 Key Management for Multi-sender Broadcast Homomorphic MACs

The key dissemination scheme described in the previous section only supports a single sender in the network. A real network however, may contain several senders. In particular, a typical node in a network may simultaneously play the role of sender, recipient and intermediate node. Supporting multiple senders in a network using the scheme outlined in the previous section requires setting up a cover free family of keys for each sender. In this section, we describe a single cover free family of keys that simultaneously supports all senders of the network.

To achieve this, we modify the key dissemination scheme as follows. Every node in the network is given two sets of keys, the first set corresponding to its role as a sender and the second set corresponding to its role as a verifier. We assume that every node in the network is identified by a sender id that is unique. We denote the sender id of node i by sid_i.

As before, the network is associated with a cover free family $(\mathbb{X}, \mathbb{B})$, where $\mathbb{X}$ is a set of keys and $\mathbb{B}$ is an ordered set of subsets of keys of $\mathbb{X}$. Let $|\mathbb{X}| = \ell$ and let the members of $\mathbb{X}$ be denoted by $x_1, x_2, \ldots, x_l$.

Given a cover free family, keys are distributed to individual nodes as follows. The node i with sender id sid_i is given two sets of keys $K_{1,i}$ and $K_{2,i}$, where $K_{1,i}$ is used to sign a message and $K_{2,i}$ is used to verify the authenticity of a received message. We let $K_{1,i} = \{F(x_1, \mathsf{sid}_i), F(x_2, \mathsf{sid}_i), \ldots,\ F(x_l, \mathsf{sid}_i)\}$ where F is a PRF. Note that $|K_{1,i}| = \ell$ for all $i = 1, \ldots, \mu$. The key $K_{2,i}$ is a block in the cover free family, i.e. $K_{2,i} \in \mathbb{B}$. Let $K_{2,i} = \{x_{i_1}, x_{i_2}, \ldots, x_{i_b}\}$ where b is the block size.

We claim that this setup is sufficient for every node in the network to play its dual roles of sender and verifier. To sign a packet, node i simply uses its ℓ keys from set $K_{1,i}$ to create ℓ tags for the packet. To verify a packet p, node i first reads the sender id of p, say sid_p (note that each packet carries with it the id of its sender). Then it uses $K_{2,i}$ to dynamically compute the b keys it needs to verify p as $\{F(x_{i_1}, \mathsf{sid}_p), F(x_{i_2}, \mathsf{sid}_p), \ldots, F(x_{i_b}, \mathsf{sid}_p)\}$. Using these b keys, the node proceeds to verify p as before. Thus, using the sender id of each received packet, a node *computes on the fly* the keys it needs for verification.

The proof of security largely remains the same as in Theorem 6. We briefly outline it here. As before, knowing a single block (or union of c blocks) of keys is not enough to fool any other node in the system because $(\mathbb{X}, \mathbb{B})$ is a (c, d) cover free family. A node i knows only b "plain" keys from $\mathbb{X}$ via its set $K_{2,i}$. For all other keys $x \in \mathbb{X}$ node i only has $F(x, \mathsf{sid}_i)$ from its key $K_{1,i}$. But since F is a secure PRF, the value $F(x, \mathsf{sid}_i)$ reveals no information about $F(x, \mathsf{sid}_j)$ for $\mathsf{sid}_j \neq \mathsf{sid}_i$.

7 Experimental Results

We implemented the homomorphic broadcast MAC outlined in Section 5 to measure its performance. In our implementation, we chose $q = 256$, i.e. we worked over the field $\mathbb{F}_{2^8}$. For brevity, we will denote this field by $\mathbb{F}$. Our messages were chosen as vectors of length 1024 over the field $\mathbb{F}$, and the network coding coefficients were picked randomly from $\mathbb{F}$. For our experiments, we restrict attention to networks with a single sender. We ran two experiments using the following two cover free families, which we constructed from polynomials [17].

- A $(2,1)$ cover-free family where $|\mathbb{X}| = 49$ and each block contains 7 keys. The number of verifiers it can support is $\mu = 7^4 = 2401$.
- A $(2,5)$ cover-free family where $|\mathbb{X}| = 121$ and each block contains 11 keys. The number of verifiers it can support is $\mu = 11^4 = 14641$.

In our implementation, we chose $m = 5$, so the sender sends 5 messages, each a 1 kilobyte vector as described above. Each message is signed with 49 (resp. 121) keys by the sender. An intermediate node (or router) receives 5 messages with 49 (resp. 121) tags each, which it linearly combines to yield an aggregate message and an aggregate tag. We verify that the resultant aggregate tag is valid for the aggregate message.

Since our homomorphic MAC requires fast multiplication in $\mathbb{F}$, we created a multiplication table offline which stores all 2^{16} products of pairs of elements of $\mathbb{F}$. This table speeds up product computation, which is now just a quick table lookup. The addition in the field is implemented as a simple XOR operation. We implemented the pseudorandom function F and the pseudorandom generator G using AES (from OpenSSL).
We timed the following operations:

1. Signing: Source signs one message.
2. Combine and Verify: Router receives five (message,tag) pairs and computes a random linear combination of the five vectors and their corresponding tags. Then it verifies that the combined tag is valid for the combined message.

The results for both cover free families are shown in the following table. Timing units are in microseconds. The numbers in the table correspond to averages taken over 10,000 runs of each experiment.

Operation timed	Sign	Combine & Verify	tag size(bytes)	Security
$p = 7$	430.3	88.5	49	$(1/2)^8$
$p = 11$	1329.3	161.5	121	$(1/2)^{40}$

These experiments were conducted on a GNU/Linux system with 4 Intel Xeon 3 Ghz processors with symmetric multiprocessing support.

8 Conclusions

We presented a homomorphic MAC suitable for networks using network coding. The homomorphic MAC can be converted to a broadcast homomorphic MAC using cover free families. The resulting broadcast MAC is collusion resistant up to a pre-determined collusion bound c. The tag size grows quadratically with c.

Our experimental results show that the MAC performs well as a point-to-point MAC. As a broadcast MAC it performs well for small values of c. It is an interesting question whether a TESLA-type mechanism [14] applied to our homomorphic MAC can be used to give the same functionality as our broadcast MAC, where every intermediate network router can verify the tag.

References

1. Ahlswede, R., Cai, N., Li, S., Yeung, R.: Network information flow. IEEE Transactions on Information Theory 46(4), 1204–1216 (2000)
2. Koetter, R.: An algebraic approach to network coding. IEEE/ACM Transactions on Networking 11, 782–795 (2003)
3. Li, S.Y.R., Yeung, R.W., Cai, N.: Linear network coding. IEEE Trans. Inform. Theory 49(2), 371–381 (2003)
4. Boneh, D., Freeman, D., Katz, J., Waters, B.: Signing a linear subspace: Signature schemes for network coding. In: Proc. of PKC 2009 (2009)
5. Zhao, F., Kalker, T., Médard, M., Han, K.: Signatures for content distribution with network coding. In: Proc. of International Symposium on Information Theory (ISIT) (2007)
6. Han, K., Ho, T., Koetter, R., Medard, M., Zhao, F.: On network coding for security. In: Military Communications Conference (Milcom) (2007)
7. Charles, D., Jain, K., Lauter, K.: Signatures for network coding. In: CISS 2006 (2006); to appear in International Journal of Information and Coding Theory
8. Krohn, M., Freedman, M., Mazieres, D.: On the-fly verification of rateless erasure codes for efficient content distribution. In: Proc. of IEEE Symposium on Security and Privacy, pp. 226–240 (2004)
9. Gkantsidis, C., Rodriguez, P.: Network coding for large scale content distribution. In: Proc. of IEEE INFOCOM 2005, pp. 2235–2245 (2005)
10. Gkantsidis, C., Rodriguez, P.: Cooperative security for network coding file distribution. In: INFOCOM (2006)
11. Gkantsidis, C., Miller, J., Rodriguez, P.: Comprehensive view of a live network coding p2p system. In: Internet Measurement Conference, pp. 177–188 (2006)
12. Carter, L., Wegman, M.: Universal classes of hash functions. Journal of Computer and System Sciences 18(2), 143–154 (1979)
13. Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., Pinkas, B.: Multicast security: A taxonomy and some efficient constructions. In: Proc. of INFOCOM 1999, vol. 2, pp. 708–716 (1999)
14. Perrig, A., Canetti, R., Tygar, D., Song, D.: Efficient authentication and signature of multicast streams over lossy channels. In: Proc. of 2000 IEEE Symposium on Security and Privacy (2000)
15. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) Asiacrypt 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)
16. Katz, J., Lindell, Y.: Introduction to Modern Cryptography: Principles and Protocols. CRC Press, Boca Raton (2007)
17. Kumar, R., Rajagopalan, S., Sahai, A.: Coding constructions for blacklisting problems without computational assumptions. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 609–623. Springer, Heidelberg (1999)

Algorithmic Tamper Proof (ATP) Counter Units for Authentication Devices Using PIN

Yuichi Komano[1], Kazuo Ohta[2], Hideyuki Miyake[1], and Atsushi Shimbo[1]

[1] Toshiba Corporation,
1, Komukai Toshiba-cho, Saiwai-ku, Kawasaki 212-8582, Japan
{yuichi1.komano,hideyuki.miyake,atsushi.shimbo}@toshiba.co.jp
[2] The University of Electro-Communications,
Chofugaoka 1-5-1, Chofu-shi, Tokyo 182-8585, Japan
ota@ice.uec.ac.jp

Abstract. Though Gennaro et al. discussed the algorithmic tamper proof (ATP) devices using the personal identification number (PIN) with less tamper-proof devices, and proposed counter units which count the number of wrong attempts in user authentication; however, as for the counter unit, they only constructed one which counts the *total* number of wrong attempts. Although large number for the limit of wrong attempts is required for usability, it allows an attacker to search PIN up to the limit and degrades the security. The construction of secure counter units which count the number of *consecutive* wrong attempts remains as an open problem. In this paper, we first formalize the ATP security of counter units, and propose two constructions of counter unit which count the number of consecutive wrong attempts. The security of each construction can be proven under the assumptions of secure signature scheme and random function. The former one is required to store two states in secure memory area (RP-Mem) with low computation cost; and the latter one has high computation cost but is required to store only one state in RP-Mem. This shows the trade-off between the costs of hardware and algorithm.

Keywords: algorithmic tamper proof (ATP), counter unit, PIN authentication.

1 Introduction

1.1 Background

Physical attacks like the fault attack [2] threaten the security of devices and several countermeasures with particular hardware are proposed. Gennaro et al. [3] discussed the algorithmic tamper proof (ATP) devices in which software algorithm cooperates with hardware to ensure the security against such attacks. It relaxes the hardware requirements and gives us practical device implementations.

In [3], they constructed the ATP devices, such as decryption and signature generation ones. Moreover, they insisted that, in order to prohibit unauthorized

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 306–323, 2009.

users from running the devices, an authentication using the personal identification number (PIN) should be performed. In order to secure the device, they claimed that it should self-destruct if more wrong attempts of authentication are detected than the limit number which is pre-determined as a system parameter.

In considering ATP devices, the property of memory area must be carefully discussed. They [3] prepare two kinds of memory area for the device. One is a *read-proof* but tamperable area (we call it *read proof memory*, RP-Mem); and the other is a *tamper-proof* but readable one (*tamper proof memory*, TP-Mem). Read-proof (resp. tamper-proof) memory is one where attacker is not permitted to read (write by fault injection etc.) data in (into) it. In order to decrease the manufacture cost, TP-Mem should be a read only memory where the common data for devices is hardwired (see [3] or section 3.2). This paper also uses RP-Mem and TP-Mem to construct ATP counter units as well as [3].

There are two types of counter unit. One counts the *total* number of wrong attempts; and the other counts the number of *consecutive* wrong ones. In [3], they gave a concrete unit for the former one and claimed its security (without a formal discussion); however, the former one has a dilemma in choosing the limit of wrong attempts. Although the limit number should be large enough in case the legal user misses the authentication during the life time of devices; it allows an attacker to search the PIN by try and error up to the limit, and degrades the security. The latter one can solve this dilemma but the ATP secure construction of such unit remains as an open problem.

Let us review the counter unit proposed in [3] for the total number of wrong attempts (see appendix for detail). The unit stores a state (R_i) in RP-Mem which is a node of hash chain and relates to the total number of wrong attempts. The leaf of hash chain is signed by the trusted party. When an attempt fails, it updates the state with hashing it (move to next node). If a certain attacker over-writes the state maliciously, then it can be detected because the signature must not be valid for a leaf of hash chain from the over-written state; and then the device self-destructs to prevent the attacker from accessing it. If the state is readable for the attacker, then it is possible to rewind the counter by copying the state; therefore, the state should be stored in RP-Mem. See appendix A.1 for detail.

1.2 Our Contribution

In this paper, we first define a security model of counter units by giving the attack scenario and goal formally. In this model, an attacker can read (write) data in (into) TP-Mem (RP-Mem); and moreover, she can run functionalities of counter units at her will. We then propose two constructions of a counter unit which counts the number of consecutive wrong attempts and prove their security.

Note that the counter unit which counts the number of consecutiv wrong attempts requires a reset functionality and our security model allows an attacker to use it. In order to prevent the attacker maliciously running this functionality, we let reset the counter after PIN is certified in the reset functionality.

The first construction, named ATP-CU1 (algorithmic tamper-proof counter unit 1), stores *two* states $state_1$ and $state_2$ (nodes of a hash chain) in RP-Mem.

$state_1$ represents the total number of wrong attempts like [3] and $state_2$ relates to the latest success of the authentication. The correctness of these states can be checked whether $state_2$ is mapped to $state_1$ with the hash function (strictly, we assume a pseudorandom function); and the number of consecutive wrong attempts is measured by the distance between them (figure 2). The security of ATP-CU1 can be ensured by the pseudorandomness of hash function and the unforgeability of underlying signature scheme.

The other one, named ATP-CU2, is based on *one* state and a chameleon hash function [6,1]. The state directly correlates to the number of consecutive wrong attempts and its correctness is checked whether the chameleon hash function maps the state to the pre-fixed data with respect to the signature stored in RP-Mem. The state is updated with the open algorithm of the chameleon hash function. ATP-CU2 is secure if the chameleon hash function is collision resistant one-way and if the underlying signature scheme is unforgeable. Compared to ATP-CU1, ATP-CU2 utilizes the chameleon hash function to make data (corresponds to $state_2$ in ATP-CU1) fixed and to decrease the variable state by one; however, the chameleon hash function generally requires high computation and long output. This shows the trade-off between the costs of hardware and algorithm.

This paper is organized as follows. Section 2 reviews the definitions of preliminaries. In section 3, we model the ATP devices using PIN with counter units and formalize ATP security of counter units. We then construct two counter units for the consecutive wrong attempts and prove their security in sections 4 and 5, respectively. Section 6 makes the discussion and section 7 concludes this paper. Moreover, in appendix A, we revisit the counter unit [3] for the total number of wrong attempts to give a formal discussion.

2 Preliminaries

Let us review the signature scheme, pseudorandom function and chameleon hash function. We first recall the model of signature scheme and its security [5] this paper assumes. Since the existential unforgeability against key only attack, *not* against chosen message attack, is enough for ensuring the security of our units, this section review the former one. We assume the *trusted* device vendor generates a pair of public and secret keys, makes signatures and stores the pulic key and signatures in *tamper-proof* and *read-proof* memories (see section 3.2), respectively; and therefore, no one except the vendor gets a pair of message and signatures and our assumption of key only attack is meaningful.

Definition 1 (Signature Scheme). The signature scheme consists of the following three algorithms.

1) **Key Generation Algorithm,** Gen**:** Given a security parameter k as an input, it outputs a key pair of public and secret keys $\mathsf{Gen}(1^k) = (pk, sk)$. This algorithm is probabilistic.

2) **Signing Algorithm,** Sign**:** Given a message m and a secret key sk as inputs, it outputs a signature $\mathsf{Sign}_{sk}(m) = s$. This algorithm may be probabilistic.
3) **Verification Algorithm,** Ver**:** Given a message mCa signature s, and a public key pk as inputs, it outputs $\mathsf{Ver}_{pk}(m, s) = 1$ (accept) if s is valid for m and pk; or $\mathsf{Ver}_{pk}(m, \sigma) = 0$ (reject) otherwise. This algorithm is deterministic.

Definition 2 (Existential Unforgeability against Key Only Attack). We say that a signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is existentially unforgeable against key only attack in (τ, ϵ) if, for all attacker A whose running time is bounded by τ, the success probability of A is at most ϵ. Here, the success probability of A is defined by $\Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^k); (m^*, s^*) \leftarrow A(pk) : \mathsf{Ver}_{pk}(m^*, s^*) = 1]$.

We then recall the definition of pseudorandom function [4].

Definition 3 (Pseudorandom Function). For a security parameter k, the family of hash functions $\{f_k : \{0,1\}^k \to \{0,1\}^k\}$ is pseudorandom if it satisfies the following property: For all polynomial time algorithm M, $\epsilon = |\Pr[M^{f_{U_k}}(1^k) = 1] - \Pr[M^{F_k}(1^k) = 1]| < \frac{1}{p(k)}$ holds for all positive function p and sufficiently large k. Here, F_k is an arbitrary function from $\{0,1\}^k$ to $\{0,1\}^k$ and U_k is an element chosen randomly and uniformly from $\{0,1\}^{|k|}$.

Finally, let us recall the definition of chameleon hash function [6,1].

Definition 4 (Chameleon Hash Function). Let R be a recipient and let k, k_m, k_r be security parameters. The chameleon hash function consists of the following three algorithms.

1) **Key Generation Algorithm,** CGen**:** Given k, it outputs a key pair of public and private keys $\mathsf{CGen}(1^k) = (CPK, CSK)$ for R. This algorithm is probabilistic.
2) **Chameleon Hash Algorithm,** CH**:** Given a message $m \in \{0,1\}^{k_m}$, a randomness $r \in \{0,1\}^{k_r}$, and CPK; it outputs a hash value $\mathsf{CH}_{CPK}(m, r) \in \{0,1\}^k$. This algorithm is deterministic.
3) **Open Algorithm** CHI**:** Given messages $m, m' \in \{0,1\}^{k_m}$, a randomness $r \in \{0,1\}^k$, and CSK; it outputs $\mathsf{CHI}_{CSK}(m, r, \mathsf{CH}_{CPK}(m, r)) = r'$ such that $\mathsf{CH}_{CPK}(m, r) = \mathsf{CH}_{CPK}(m', r')$ holds.

Definition 5 (Properties of CH**).** Let R be a recipient and let CPK and CSK denote the public and secret keys of R, respectively. We say that the chameleon hash function CH_{CPK} (and its open algorithm CHI_{CSK}) is secure if it satisfies the following properties.

1) **Uniformity:** For all m, if r is chosen at random from $\{0,1\}^{k_r}$, then $\mathsf{CH}_{CPK}(m, r)$ is uniformly distributed in $\{0,1\}^k$.
2) **Collision Resistance:** We say that the chameleon hash function CH_{CPK} is collision resistant in (τ_C, ϵ_C) if, for all attacker A whose running time is bounded by τ_C, the success probability of A is at most ϵ_C. Here, the success probability of A is defined by $\Pr[(CPK, CSK) \leftarrow \mathsf{CGen}(1^k); \{(m, r), (m', r')\} \leftarrow A(CPK) : (m \neq m') \wedge (\mathsf{CH}_{CPK}(m, r) = \mathsf{CH}_{CPK}(m', r'))]$.

3 ATP Devices Using PIN

This section formalizes the ATP devices which authorize a user with a PIN. We assume that no one can guess PIN with success probability more than ϵ_P within time bound τ_P. Note that ϵ_P is polynomial, *e.g.*, $\frac{1}{10^4}$ for four digit PIN; however, we assume that it has enough entropy (*i.e.*, PIN is chosen randomly) for disturbing a few guesses allowed by devices with our counter unit.

3.1 Function of Devices

This paper deals with devices which self-destruct if the authentication fails more than the limit number (*e.g.*, 3 times), in order to prevent malicious users from using functionalities of such devices. For this purpose, a counter unit which counts the number of wrong attempts in authentication should be implemented. See figure 1 for its construction.

There are two types of functionalities of devices. One consists of sensitive functionalities that should be invoked only after the authentication succeeds (*e.g.*, signing and decryption functionalities); the other consists of ones which can be called at any time (*e.g.*, authentication functionality).

The device works in the following way. The counter is initialized with zero. A user who requests to use a sensitive functionality first runs the authentication functionality. If the authentication succeeds, then she is allowed to run the sensitive one. If not, on the other hand, the device increments the counter with update functionality (subroutine of authentication) and rejects her request.

3.2 Memory Area of Devices

We classify the memory unit of device into four types with respect to readability and tamperability for an attacker as in Table 1. In this table, NP-Mem stands for a non-protected memory and an attacker can read and write data in NP-Mem by probing etc. RP-Mem is a read-proof memory where an attacker can write data into it by injecting faults etc., but can not read data in it. It can be realized by the

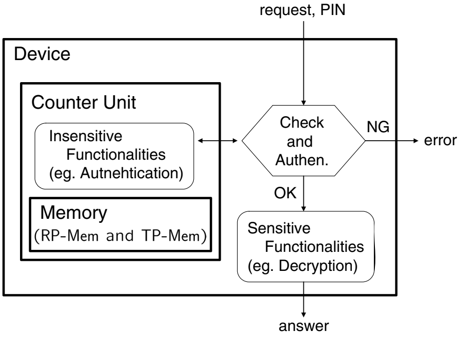

request, PIN
Device
Counter Unit
Insensitive Functionalities (eg. Autnehtication)
Memory (RP-Mem and TP-Mem)
Check and Authen.
NG
error
OK
Sensitive Functionalities (eg. Decryption)
answer

Fig. 1. Devices with Counter Units

Table 1. Types of Memory Unit

		Read	
		Yes	No
Write	Yes	NP-Mem	RP-Mem
	No	TP-Mem	RTP-Mem

special hardware [7] or memory encryption[1], for example. TP-Mem is a tamper-proof memory where an attacker can read data in it but can not write data into it. From the viewpoint of mass production [3], TP-Mem stores (hardwires) the common data for devices and is regarded as a read only memory (ROM). Our aim is, following that of [3], to implement a secure counter unit with RP-Mem and TP-Mem (and NP-Mem for storing temporary data).

Note that, if the read-proof and tamper-proof area (RTP-Mem in Table 1) is available, the secure counter unit can be obviously constructed when RTP-Mem holds the number of wrong attempts. However, RTP-Mem costs high and seems not to be implemented in small devices; therefore, we follow [3] to assume RTP-Mem is not available. Note also that, since memory area of RP-Mem and TP-Mem tends to highly cost compared to NP-Mem, we should estimate the allocation for each area and decrease the use of area highly costing.

3.3 Security Requirements for ATP Devices Using PIN

In this subsection, we model the ability (attack scenario) of attacker against the device and her aim (attack goal).

Ability of Attacker. We assume that an attacker can invoke insensitive functionalities at will. In this scenario, the device can be utilized as a black box to give her hints. Moreover, she is allowed to change the data stored in tamperable memory areas (NP-Mem and RP-Mem) and to read the data stored in readable memory areas (NP-Mem and TP-Mem) on her will.

This scenario is an extension of the fault attack to cryptographic devices [2]. Note that while the practical fault attacks change the data by chance; our scenario allows her to specify the position and value of data flipped by her attack. She is, however, allowed neither to run the sensitive functionality directly (bypassing the authentication) nor to skip the operation inside the insensitive functionality[2].

[1] Devices encrypt the data and store the encrypted one. The devices hold a decryption key for read the data. In practice, the memory encryption seems a practical but the decryption key should be managed properly. In order to discuss the provable security, if the memory encryption is utilized to realize RP-Mem, the key management is also modeled formally; however, it is an out of scope how RP-Mem is realized for this paper, we omit it.

[2] See Remark 1 in appendix A for the necessity of this assumption.

We give the attack model with following queries.

- *data read query:* With the query $\mathsf{Read}(\mathsf{type}, \mathsf{var})$, the attacker receives the value val of the variable var if it is stored in $\mathsf{type} \in \{\mathsf{NP\text{-}Mem}, \mathsf{TP\text{-}Mem}\}$; or $\perp$ otherwise. Assume that the output is returned immediately after the query.
- *data write query:* With the query $\mathsf{Write}(\mathsf{type}, \mathsf{var}, h)$, the attacker replaces the value val with $h(\mathsf{val})$ of the variable var if it is stored in $\mathsf{type} \in \{\mathsf{NP\text{-}Mem}, \mathsf{RP\text{-}Mem}\}$; or receives $\perp$ otherwise. Assume that the replacement (or the output of $\perp$) is performed immediately after the query.
- *functionality run query:* With the query $\mathsf{Run}(\mathsf{func}, \mathsf{ope})$, the attacker runs the functionality func with the operand ope and receives its output if it exists. Assume that, while some functionality runs with this query, he is allowed to request data read and write queries but she is disallowed to request another functionality run query concurrently[3].

Note that functionalities performed by the functionality run query above vary from units. For instance, ATP-CU1 has three functionalities (Main, Update_1, and Update_2) and we allow an attacker to run them with the query.

Goal of Attacker. Although the supreme goal of an attacker is to recover PIN, we impose her with another goal: to attempt the authentication more than the limit. For example, assume that the limit number is three. In this setting, the device should self-destruct if fourth wrong attempt happens (in total or consecutive). We say that an attacker succeeds in this attack if she performs fifth attempt with no knowledge about PIN. Note that, if this attack is mounted, there is a chance to find PIN by try and error.

ATP Security for Devices with Counter Unit. We call the device with counter unit is ATP secure if there is no polynomial time attacker who achieves the attack goal under the attack scenarios above with non-negligible success probability.

4 Algorithmic Tamper-Proof Counter Unit 1: ATP-CU1

This section shows the counter unit ATP-CU1 which counts the number of consecutive wrong attempts. This is one of solutions to the open problem of [3].

4.1 Construction of ATP-CU1

ATP-CU1 utilizes two states $state_1 = R$ and $state_2 = S$ which are the nodes of a hash chain (strictly, a chain of pseudorandom function). R represents the total number of wrong attempts as in [3] and S relates to the latest success attempt. Their correctness can be checked whether S is mapped to R with the hash function G; $R = \mathsf{G}^i(\sigma_1, S)$ for $i \leq m$ where σ_1 and m are the signature

[3] See Remark 2 in appendix A for the necessity of this assumption.

Table 2. Memory Allocation for ATP-CU1

TP-Mem	$m, k, \mathsf{G}, \mathsf{Ver}, pk$
RP-Mem	$state_1 = R, state_2 = S, \sigma_1, \sigma_2$

and limit number, respectively (see below). If they are correct, the number of consecutive wrong attempts is measured by the distance between them; i in the previous equality. If the attempt fails, it updates R by replacing it with $\mathsf{G}(\sigma_1, R)$. On the other hand, if the attempt succeeds, it updates $state_2 = S$ by replacing it with $\mathsf{G}^{i-1}(\sigma_1, S)$. See figure 2 for example.

Notation. We introduce the notations of variables and preliminaries for ATP-CU1.

- PIN: PIN
- m: limit number of wrong attempts
- k: security parameter
- $\{\mathsf{G}(\sigma, \cdot) : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k\}$: pseudorandom function[4]
- $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$: signature scheme
- (pk, sk): public and secret keys for signature scheme with respect to the trusted device vendor[5]
- $state_1 = R$ and $state_2 = S$

The vendor runs $\mathsf{Gen}(1^k)$ to generate pk and sk that are used for all devices in advance.

Initialization. The vendor initializes the device as follows.

1. He decides PIN (if necessary, by interacting with a user).
2. He computes $\sigma_1 \leftarrow \mathsf{Sign}_{sk}(PIN)$ and $\sigma_2 \leftarrow \mathsf{Sign}_{sk}(\sigma_1)$.
3. He chooses $S \in \{0,1\}^k$ at random.
4. He computes $R \leftarrow \mathsf{G}(\sigma_1, S)$.
5. He initializes $state_1 \leftarrow R$ and $state_2 \leftarrow S$.
6. He allocates the data as in Table 2.

Note that at step 2, PIN may be padded with a random nonce (in this case, it is stored in RP-Mem and used in verification) in order to enlarge the entropy of the input of signing function.

The authentication (check for the correctness of inputted PIN') is performed by the verification of PIN' and σ_1. Moreover, σ_1, which is a part of input of G, disable an attacker to predict the output of G. On the other hand, σ_2 ensures that σ_1 (stored in tamperable area) is unchanged by $\mathsf{Write}(\mathsf{RP\text{-}Mem}, \sigma_1, h)$ with signing verification. This verification ensures that σ_2 is also unchanged.

We denote, by $\mathsf{G}^n(\sigma_1, R)$ for $\sigma_1 \in \{0,1\}^k$ and $R \in \{0,1\}^k$, the operation where G is repeatedly applied n times. For instance, $\mathsf{G}^2(\sigma_1, R) = \mathsf{G}(\sigma_1, \mathsf{G}(\sigma_1, R))$.

[4] Assume that the first input of G (signature σ_1 below) is a key for pseudorandom function.

[5] Hereafter, we assume that the device vendor is trusted.

Authentication Process. In correspondence with a request from a user, the device authorizes the user and updates the states as follows. Figure 2 shows the translation of states utilized in ATP-CU1.

—— $\mathsf{Main}(PIN')$ ——

1. If $R \notin \{\mathsf{G}(\sigma_1, S), \cdots, \mathsf{G}^{m+1}(\sigma_1, S)\}$ holds, then it self-destructs.
2. If $\mathsf{Ver}_{pk}(\sigma_1, \sigma_2) = 0$ holds, then it self-destructs.
3. Otherwise, the following steps are performed.
   (a) If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 1$ holds, then the following steps are performed.
      i. If $\mathsf{G}(\sigma_1, S) = R$ holds, then it accepts the request.
      ii. If $\mathsf{G}(\sigma_1, S) \neq R$ holds, then it runs $\mathsf{Update}_2(PIN')$ and accepts the request.
   (b) If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 0$ holds, then it runs $\mathsf{Update}_1(\phi)$ and rejects the request.

—— $\mathsf{Update}_1(\phi)$ ——

1. If $\mathsf{Ver}_{pk}(\sigma_1, \sigma_2) = 0$ holds, then it self-destructs.
2. It updates R with $\mathsf{G}(\sigma_1, R)$.

—— $\mathsf{Update}_2(PIN')$ ——

1. If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 0$ holds, then it self-destructs.
2. If $\mathsf{Ver}_{pk}(\sigma_1, \sigma_2) = 0$ holds, then it self-destructs.
3. Otherwise, the folloing steps are performed.
   (a) If $R = \mathsf{G}^n(\sigma_1, S) \in \{\mathsf{G}^2(\sigma_1, S), \cdots, \mathsf{G}^{m+1}\ (\sigma_1, S)\}$ holds, then it updates S with $\mathsf{G}^{n-1}(\sigma_1, S)$.
   (b) If $R \notin \{\mathsf{G}^2(\sigma_1, S), \cdots, \mathsf{G}^{m+1}(\sigma_1, S)\}$ holds, then it self-destructs.

The first step of Main computes the hash chain for each invocation and checks the number of consecutive wrong attempts is within the limit.

Note that the number of consecutive wrong attempt is estimated by $n - 1$ if $R = \mathsf{G}^n(\sigma_1, S)$ holds at step 1 in Main. Also note that we assume that an attacker can run the subroutines of authentication (Update_1 and Update_2) directly. Therefore, in order to prevent the attacker from running such subroutines maliciously, the checks performed in Main are also done in Update_1 and Update_2.

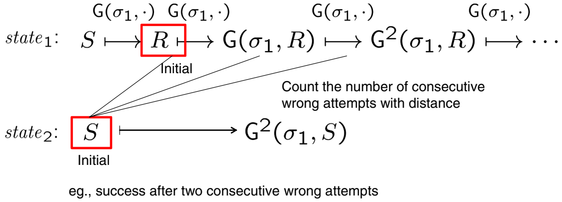

$\mathsf{G}(\sigma_1, \cdot)$
$\mathsf{G}(\sigma_1, \cdot)$
$\mathsf{G}(\sigma_1, \cdot)$
$\mathsf{G}(\sigma_1, \cdot)$
$state_1$: $S \longmapsto R \longmapsto \mathsf{G}(\sigma_1, R) \longmapsto \mathsf{G}^2(\sigma_1, R) \longmapsto \cdots$
Initial
Count the number of consecutive wrong attempts with distance
$state_2$: $S \longmapsto \mathsf{G}^2(\sigma_1, S)$
Initial
eg., success after two consecutive wrong attempts

Fig. 2. Translation of States in ATP-CU1

4.2 Security Consideration of ATP-CU1

As for the security of ATP-CU1, we will prove the following theorem.

Theorem 1. If it is infeasible to find *PIN* within time bound τ_P and with the probability more than ϵ_P, if it is infeasible to break the pseudorandomness of G within time bound τ_G with the probability more than ϵ_G, and if it is infeasible to break existential unforgeability against key only attack within time bound τ_B with the probability ϵ_B, then it is infeasible to break the ATP security of ATP-CU1 within time bound τ_A with the probability more than ϵ_A. The following inequalities hold.

$$\begin{cases} \tau_B \leq (m+2)\tau_P + \tau_G + \tau_A + O(1)\mathsf{T}, \\ \epsilon_B \geq \epsilon_A - (m+2)\epsilon_P - \epsilon_G - 1/2^{k-1} \end{cases}$$

Here, T denotes the running time of authentication device once.

Proof: By using the attacker A against ATP-CU1 as a black box, we construct an algorithm B which takes the key only attack to existentially forge a signature of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$.

The aim of A is to perform the PIN authentication $m+2$ times. As described in section 3.3, A outputs the following queries: data read query $\mathsf{Read}(\mathsf{type}, \mathsf{var})$, data write query $\mathsf{Write}(\mathsf{type}, \mathsf{var}, h)$, and functionality run query $\mathsf{Run}(\mathsf{func}, \mathsf{ope})$ where $\mathsf{func} \in \{\mathsf{Main}, \mathsf{Update}_1, \mathsf{Update}_2\}$.

B, given a public key pk of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ and a pseudorandom function G as inputs, simulates the answers to queries from A and uses A as a black box to output a forgery (m^*, s^*) of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$. B first initializes the input of A and data for a device as follows.

Initialization:

1. B decides *PIN*.
2. B chooses $\sigma_1, \sigma_2 \in \{0,1\}^k$ at random.
3. If $\mathsf{Ver}_{pk}(PIN, \sigma_1) = 1$ holds, then B outputs (PIN, σ_1) as a forgery and stops.
4. If $\mathsf{Ver}_{pk}(\sigma_1, \sigma_2) = 1$ holds, then B outputs (σ_1, σ_2) as a forgery and stops.
5. B chooses $S \in \{0,1\}^k$ at random.
6. B computes $R \leftarrow \mathsf{G}(\sigma_1, S)$.
7. B sets $state_1 = R$ and $state_2 = S$.

B invokes A with inputs G and pk, and answers to queries from A as follows.

Answers to $\mathsf{Read}(\mathsf{type}, \mathsf{var})$*:*

1. If $\mathsf{type} \notin \{\mathsf{NP\text{-}Mem}, \mathsf{TP\text{-}Mem}\}$ holds, then B returns $\perp$ to A.
2. B returns the value val of the variable var to A.

Answers to $\mathsf{Write}(\mathsf{type}, \mathsf{var}, h)$*:*

1. If $\mathsf{type} \notin \{\mathsf{NP\text{-}Mem}, \mathsf{RP\text{-}Mem}\}$ holds, then B returns $\perp$ to A.
2. B replaces the value val of the variable var with $h(\mathsf{val})$.

Answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$*:*

1. If another $\mathsf{Run}(*, *)$ is being performed, then B returns $\perp$ to A.
2. If A replaces at least one of σ_1 and σ_2 by different value with Write query, then the following steps are performed.
   (a) Same as step 3 of Initialization.
   (b) Same as step 4 of Initialization.
   (c) Otherwise, B halts (this case corresponds to the self-destruction).
3. If A does not replace σ_1 nor σ_2 by different value with Write query, then B updates R with $\mathsf{G}(\sigma_1, R)$.

Answers to $\mathsf{Run}(\mathsf{Update}_2, \mathit{PIN}')$*:*

1. Same as step 1 of answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$.
2. If $\mathsf{Ver}_{pk}(\mathit{PIN}', \sigma_1) = 1$ holds, then B outputs $(\mathit{PIN}', \sigma_1)$ as a forgery and stops.
3. Same as step 2 of answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$.
4. If A does not replace σ_1 nor σ_2 by different value with Write query, then B terminates and fails to output a forgery (B aborts).

Answers to $\mathsf{Run}(\mathsf{Main}, \mathit{PIN}')$*:*

1. Same as step 1 of answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$.
2. If $R \notin \{\mathsf{G}(\sigma_1, S), \cdots, \mathsf{G}^{m+1}(\sigma_1, S)\}$ holds, then B halts.
3. Same as step 2 of answers to $\mathsf{Run}(\mathsf{Update}_2, \mathit{PIN}')$.
4. Same as step 2 of answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$.
5. If this is the $(m + 2)$-th run for $\mathsf{Run}(\mathsf{Main}, *)$ and if A does not replace σ_1 by different value with Write query, then B aborts.
6. Otherwise (before the $(m + 2)$-th run and none of σ_1 and σ_2 is replaced by different value), the following steps are performed.
   (a) If $\mathit{PIN}' = \mathit{PIN}$ holds, then B aborts.
   (b) If $\mathit{PIN}' \neq \mathit{PIN}$ holds, then B runs $\mathsf{Update}_1(\phi)$ and rejects the request.

In the following three cases, B fails to simulate the answers and to output a forgery (*i.e.*, B aborts): step 4 of answers to $\mathsf{Run}(\mathsf{Update}_2, \mathit{PIN}')$, and steps 5 and 6(a) of answers $\mathsf{Run}(\mathsf{Main}, \mathit{PIN}')$. We should show that these cases occur by accident.

As for the case in step 4 of answers to $\mathsf{Run}(\mathsf{Update}_2, \mathit{PIN}')$, A can distinguish B from ATP-CU1 only when A inputs $\mathit{PIN}' = \mathit{PIN}$ but B aborts. It happens with probability less than ϵ_P.

We then estimate the probability with which B aborts in step 5 of answers to $\mathsf{Run}(\mathsf{Main}, \mathit{PIN}')$. Note that σ_1 (which is unchanged with Write query) is randomly chosen from $\{0, 1\}^k$ and stored in $\mathsf{RP\text{-}Mem}$. In this case, unless A successfully guesses σ_1 with probability $1/2^k$, σ_1 is uniformly distributed in $\{0, 1\}^k$ for A. In the $(m+2)$-th run of Main, in order to pass $R \in \{\mathsf{G}(\sigma_1, S), \cdots, \mathsf{G}^{m+1}(\sigma_1, S)\}$ (step 2) for $\mathit{state}_1 = R$ and $\mathit{state}_2 = S$, at least one of R and S should be replaced by different value with Write query. Under the assumption of uniformity of σ_1

for A, however, the output of $\mathsf{G}(\sigma_1, \cdot)$ is also uniformly distributed in $\{0,1\}^k$, unless A breaks the pseudorandomness of G with probability ϵ_G. Namely, the probability with which A replaces R and/or S to pass the check of step 2 is bounded by $1/2^k$. Therefore, A can distinguish B from ATP-CU1 in this step with probability less than $\epsilon_G + 1/2^{k-1}$.

In step 6(a) of answers to $\mathsf{Run}(\mathsf{Main}, PIN')$, A can distinguish B from ATP-CU1 with probability less than $(m+1)\epsilon_P$, because A is allowed to attempt the authentication at most $m+1$ times. ■

5 Algorithmic Tamper-Proof Counter Unit 2: ATP-CU2

We then give another construction of counter unit ATP-CU2 which also counts the number of consecutive wrong attempts and discuss its security.

5.1 Construction of ATP-CU2

ATP-CU2 utilizes a chameleon hash function CH [6,1] to update the state $state = (i, R_i)$ which corresponds to the number of consecutive wrong attempts i. The correctness of (i, R_i) is checked whether the chameleon hash function maps (i, R_i) to the pre-fixed data V with respect to the signature σ_2 stored in RP-Mem; it checks whether $V = \mathsf{CH}(\sigma_1, i, R_i)$ and σ_2 pass the signature verification or not. The state is updated with the open algorithm CHI of the chameleon hash function in order for the state to be mapped to the pre-fix data by the chameleon hash function; it finds R_j such that $\mathsf{CH}(\sigma_1, i, R_i) = V = \mathsf{CH}(\sigma_1, j, R_j)$ where $j = 0$ if the attempt succeeds and $j = i+1$ otherwise. See figure 3 for example.

Notation. In addition to PIN, m, k, $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, and (pk, sk) described in section 4.1, the following notations are used.

- $(\mathsf{CGen}, \mathsf{CH}, \mathsf{CHI})$: chameleon hash function[6] where CH_{CPK} is $\{0,1\}^{k_{m_s}} \times \{0,1\}^{k_{m_i}} \times \{0,1\}^{k_r} \rightarrow \{0,1\}^k$ and CHI_{CSK} is $\{0,1\}^{k_{m_s}} \times \{0,1\}^{k_{m_i}} \times \{0,1\}^{k_r} \times \{0,1\}^{k_{m_s}} \times \{0,1\}^{k_{m_i}} \times \{0,1\}^k \rightarrow \{0,1\}^{k_r}$, respectively
- (CPK, CSK): public and secret keys for chameleon hash function with respect to the device vendor
- $state = (i, R_i)$: state $(i \geq 0)$

The vendor runs Gen and CGen to generate (pk, sk) and (CPK, CSK) that are used for all devices in advance. Note that CH_{CPK} is collision resistant one-way and has the following open property. With the secret key CSK, CHI_{CSK} finds another preimage of $V = \mathsf{CH}_{CPK}(\sigma_1, i, R_i)$ with different $j (\neq i)$; $\mathsf{CHI}_{CSK}(\sigma_1, i, R_i, \sigma_1, j, V)$ returns R_j such that $\mathsf{CH}_{CPK}(\sigma_1, j, R_j) = V$ holds.

[6] We assume that $k_{m_s} + k_{m_i} = k_m$ holds and that the first two (and last two moreover) inputs for CH (CHI) are contatenated into k_m bit string.

Table 3. Memory Allocation for ATP-CU2

TP-Mem	$m, k, \mathsf{CH}, \mathsf{CHI}, CPK, \mathsf{Ver}, pk$
RP-Mem	$state = (i, R_i), CSK, \sigma_1, \sigma_2$

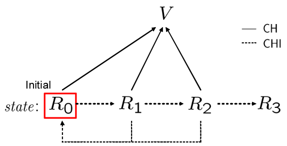


Fig. 3. Translation of State in ATP-CU2

Initialization. The vendor initializes the device as follows.

1. He decides PIN (if necessary, by interacting with a user).
2. He chooses $R_0 \in \{0,1\}^k$ at random.
3. He computes $\sigma_1 \leftarrow \mathsf{Sign}_{sk}(PIN)$; $V \leftarrow \mathsf{CH}_{CPK}(\sigma_1, 0, R_0)$; and $\sigma_2 \leftarrow \mathsf{Sign}_{sk}(V)$.
4. He initializes $state = (i, R_i) \leftarrow (0, R_0)$.
5. He allocates the data as in Table 3.

Authentication Process. In correspondence with a request from a user, the device authorizes the user and updates the states as follows. Figure 3 shows the translation of state utilized in ATP-CU2.

—— $\mathsf{Main}(PIN')$ ——
1. If $\mathsf{Ver}_{pk}(\mathsf{CH}_{CPK}(\sigma_1, i, R_i), \sigma_2) = 0$ holds, then it self-destructs.
2. Otherwise, the following steps are performed.
   (a) If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 1$ holds, then the following steps are performed.
      i. If $i > 0$ holds, then it runs $\mathsf{Update}_4(PIN')$ and accepts the request.
      ii. If $i = 0$ holds, then it accepts the request.
   (b) If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 0$ holds, then it runs $\mathsf{Update}_3(\phi)$ and rejects the request.

—— $\mathsf{Update}_3(\phi)$ ——
1. If $\mathsf{Ver}_{pk}(\mathsf{CH}_{CPK}(\sigma_1, 0, R_i), \sigma_2) = \cdots = \mathsf{Ver}_{pk}(\mathsf{CH}_{CPK}(\sigma_1,\ m, R_i), \sigma_2) = 0$ holds, then it self-destructs.
2. Otherwise, it updates (i, R_i) with $(i+1, \mathsf{CHI}_{CSK}(\sigma_1, i, R_i, \sigma_1, i+1, \mathsf{CH}_{CPK}(\sigma_1,\ i, R_i)))$.

—— $\mathsf{Update}_4(PIN')$ ——
1. If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 0$ holds, then it self-destructs.
2. If $\mathsf{Ver}_{pk}(\mathsf{CH}_{CPK}(\sigma_1, i, R_i), \sigma_2) = 0$ holds, then it self-destructs.
3. It updates (i, R_i) with $(0, R_0 \leftarrow \mathsf{CHI}_{CSK}(\sigma_1, i, R_i, \sigma_1, 0, \mathsf{CH}_{CPK}(\sigma_1, i, R_i)))$.

If the number of consecutive wrong attempts exceeds m, the device self-destructs and does not work any more at the first stem of Update_3.

5.2 Security Consideration of ATP-CU2

The security of ATP-CU2 can be similarly discussed as in section 4.2; and therefore, we omit the detail here. With regard to the security of ATP-CU2, the following theorem holds.

Theorem 2. If it is infeasible to find PIN within time bound τ_P with probability more than ϵ_P, if it is infeasible to break the security (collision resistance) of CH within time bound τ_C with probability more than ϵ_C, and if it is infeasible to break the existentially forgeability against key only attack, then it is infeasible to break the ATP security of ATP-CU2 within time bound τ_A with probability more than ϵ_A. The following inequalities hold.

$$\begin{cases} \tau_B \leq (m+2)\tau_P + \tau_G + \tau_A + O(1)\mathsf{T}, \\ \epsilon_B \geq \epsilon_A - (m+2)\epsilon_P - \epsilon_G - m/2^{k-1} \end{cases}$$

Here, T denotes the running time of authentication device once.

6 Discussion

This section first considers a naive construction of counter unit which counts the number of consecutive wrong attempts from the counter unit of [3] which counts the total number of wrong attempts. After that, let us compare the counter units; that of [3], ATP-CU1 and ATP-CU2.

6.1 Naive Construction and Its Weakness

Let us consider a counter unit which counts the number of consecutive wrong attempts with states $state_1 = (i, R)$ and $state_2 = (j, S)$ (initially, we set $i = 1$ and $j = 0$) in the same manner as the counter unit of [3]. Its algorithm is almost same as that of ATP-CU1 except it checks whether $i - j \in [1, m+1]$ and $R = \mathsf{G}^{i-j}(\sigma_1, S)$ hold instead of step 1 of $\mathsf{Main}(PIN')$ and step 3(a) of $\mathsf{Update}_2(PIN')$ for ATP-CU1.

In order to prove the security of above naive construction, we require another restriction to an attacker as follows: She is disallowed to make a data write query between the check of $i - j \in [1, m+1]$ and $R = \mathsf{G}^{i-j}(\sigma_1, S)$ is performed. Note that since i and j correlate the number of wrong attempts, she can guess them. If she is allowed to make the query between the check, the following attack can be done. (1) Before the check of $i - j \in [1, m]$, she changes $i = 1$ and $j = 0$ with data write query to pass the check, and (2) after the check of $i - j \in [1, m]$, she rewrite i and j with guessed data with data write query to pass the check of $R = \mathsf{G}^{i-j}(\sigma_1, S)$.

With the above restriction, the naive construction can be proven to be ATP secure from the similar discussion of the ATP security of ATP-CU1. On the other hand, ATP-CU1 removes the restriction by checking, with high computation cost, whether one of $R = \mathsf{G}(\sigma_1, S)$, $R = \mathsf{G}^2(\sigma_1, S)$, $\cdots$, or $R = \mathsf{G}^{m+1}(\sigma_1, S)$ holds.

6.2 Comparison

This section compares the constructions of counter units. Note that the unit of [3] has to use large m in case of mistakes of legal users; however, in order to prevent attackers from searching PIN by try and error, m cannot be enlarged so enough. On the other hand, the proposed units can use small m with keeping high security if we assume that the legal user cannot input wrong PIN consecutively so many times.

Let us consider the proposed ones. Note that the computation of chameleon hash function (CH and CHI) requires higher cost than that of ordinal hash function (G). Therefore, ATP-CU1 can require lower cost than ATP-CU2; however, ATP-CU1 should store and update two states in $\mathsf{RP\text{-}Mem}$ in secure way. This seems a trade-off between the memory use and the computation cost.

7 Conclusion

This paper is first one to model the ATP security of counter unit and to construct two ATP secure counter units counting the number of consecutive wrong attempts. The security of the first one is proven in the assumption of existentially unforgeable signature scheme against key only attack and of pseudorandom function. Moreover, the security of the second one is also proven in the assumption of existentially unforgeable signature scheme against key only attack and of secure chameleon hash function. The first one is required to store two states into $\mathsf{RP\text{-}Mem}$ but has less computation (with hash function); on the other hand, the second one needs high computation (with chameleon hash function) but is required to store one state into $\mathsf{RP\text{-}Mem}$. This paper revisits the counter unit proposed in [3] counting the total number of wrong attempts in authentication with PIN in appendix.

References

1. Ateniese, G., de Medeiros, B.: On the key exposure problem in chameleon hashes. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 165–179. Springer, Heidelberg (2005)
2. 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)
3. Gennaro, R., Lysyanskaya, A., Malkin, T., Micali, S., Rabin, T.: Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 258–277. Springer, Heidelberg (2004)
4. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. In: 25th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1984), pp. 464–479. IEEE, Los Alamitos (1984)
5. Goldwasser, S., Micali, S., Rivest, R.: A digital signature scheme against adaptive chosen message attack. Journal of Computing (Society for Industrial and Applied Mathematics) 17(2), 281–308 (1988)

6. Krawczyk, H., Rabin, T.: Chameleon signatures. In: Network and Distributed System Security Symposium, NDSS 2000. The Internet Society (2000)
7. Tuyls, P., Schrijen, G.-J., Škorić, B., van Geloven, J., Verhaegh, N., Wolters, R.: Read-proof hardware from protective coatings. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 369–383. Springer, Heidelberg (2006)

A Counter Unit of Gennaro et al. [3]

Gennaro et al. [3] constructed the counter units for the total number of wrong attempts and for the number of consecutive wrong attempts, respectively. Since the construction of latter one is not described in detail ([3] only claims that the unit uses the modulus number), this section discusses the former one only. As for the former one, however, reference [3] did not specify its algorithm nor the memory allocation for each data. This section takes form its construction.

A.1 Necessity of RP-Mem

They [3] did not mention the necessity of RP-Mem[7]. We show here that their counter unit is vulnerable when RP-Mem is not utilized. Without RP-Mem, data is stored in readable area, TP-Mem or NP-Mem.

The counter unit of [3] prepares the state R_0 (the following subsection gives notations in detail), hash chain $R_i = \mathsf{H}(R_{i-1})$ for $i = 1, 2, \cdots, m+1$, and signature σ for R_{m+1}. The state is initially set with R_0 and updated by applying H when an authentication is failed. The correctness of state R_i is checked by the verification of $\mathsf{H}^{m+1-i}(R_i)$ and σ.

Assume the device with TP-Mem and NP-Mem, without RP-Mem. In order to be updated (re-stored), the state should not be stored in TP-Mem but in NP-Mem. In this setting, the attacker A first read R_0 and copy it in local area (like his computer). When A fails in attempt, R_0 in the device is updated to $R_1 = \mathsf{H}(R_0)$. Since R_1 is stored in NP-Mem, A can replace it with R_0 (stored in local area) and try attempt infinitely.

Therefore, the read-proof (but tamperable) area, RP-Mem, should be required for storing and updating the state.

A.2 Construction of Counter Unit of [3]

A state *state* is used for counting the total number of wrong attempts. *state* is updated and unchanged when an authentication fails and succeeds, respectively.

Notation. In addition to PIN, m, k, $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, and (pk, sk) described in section 4.1, the following notations are used.

- $\mathsf{H} : \{0,1\}^k \rightarrow \{0,1\}^k$: collision resistant one-way function
- $state = (i, R_i)$: state $(i \geq 0)$

[7] TP-Mem should be required for storing the code of hash function H.

Table 4. Memory Allocation for Counter Unit of [3]

TP-Mem	$m, k, \mathsf{H}, \mathsf{Ver}, pk$
RP-Mem	$state = (i, R_i), R_{m+1}, \sigma$

Initialization. The vendor initializes the device as follows.

1. He decides PIN (if necessary, by interacting with a user).
2. He chooses $R_0 \in \{0,1\}^k$ at random.
3. He computes $R_{m+1} \leftarrow \mathsf{H}^{m+1}(R_0)$.
4. He computes $\sigma \leftarrow \mathsf{Sign}_{sk}(PIN, R_{m+1})$.
5. He initializes $state = (i, R_i) \leftarrow (0, R_0)$.
6. He allocates the data as in Table 4.

We denote, by $\mathsf{H}^n(R)$ for $R \in \{0,1\}^k$, the operation where H is repeatedly applied n times. For instance, $R_{l+2} = \mathsf{H}^2(R_l) = \mathsf{H}(\mathsf{H}(R_l))$.

In table 4, R_{m+1} and σ is stored in RP-Mem. If R_{m+1} and σ (namely, R_0) are identical for all devices, they may be stored in TP-Mem.

Authentication Process. In correspondence with a request from a user, the device authorizes the user and updates the state as follows.

—— Main(PIN') ——

1. If $\mathsf{H}^{m+1-i}(R_i) \neq R_{m+1}$ holds, then it self-destructs.
2. Otherwise, the following steps are performed.
   (a) If $\mathsf{Ver}_{pk}(PIN', \mathsf{H}^{m+1-i}(R_i), \sigma) = 1$ holds, then it accepts the request.
   (b) Otherwise, it runs Update(ϕ) and rejects the request.

—— Update(ϕ) ——

1. It updates $(i, R_i) \leftarrow (i+1, \mathsf{H}(R_i))$.

A.3 Security of Counter Unit of [3]

The reference [3] claims, without a proof, that the unit above is ATP secure if the signature scheme is existentially unforgeable [5] and if the hash function is collision resistant one-way. This subsection discusses its security. We have following three remarks.

Remark 1. If an attacker is allowed to skip the operations inside the insensitive functionality (see the third paragraph of section 3.3), she can mount an attack to the counter unit [3] as follows: She makes the unit skip step 2(b) of Main (or first step of Update) not to update *state*, she can attempt to authenticate more than the limit number. Therefore, we assume that the attacker is disallowed to skip the operations.

Remark 2. If an attacker can run functionalities concurrently with functionality run queries (see the definition of functionality run query in section 3.3), she can also mount an attack to the counter unit [3]. For instance, if she makes another function run query before the unit proceeds the step 2(b) of Main (or first step of Update), she can attempt to authenticate more than the limit number. Hence, we restrict the attacker not to make functionality run queries concurrently.

Remark 3. Note that in order to achieve its security, the signature scheme should be utilized once (*i.e.*, one-time signature scheme) and resulting signature should be stored in RP-Mem. This is because, if an attacker can obtain or read a signature corresponding some *PIN*, she may rewind the counter with the signature and exhaustively search *PIN*.

With regard to the security of their unit, the following theorem holds.

Theorem 3. If it is infeasible to find *PIN* within time bound τ_P with probability more than ϵ_P, if it is infeasible to break the pseudorandomness of H within time bound τ_H with probability more than ϵ_H, and if it is infeasible to break the existentially forgeability of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ against key only attack, then it is infeasible to break the ATP security of their unit within time bound τ_A with probability more than ϵ_A. The following inequalities hold.

$$\begin{cases} \tau_B \leq (m+2)\tau_P + \tau_H + \tau_A + O(1)\mathsf{T}, \\ \epsilon_B \geq \epsilon_A - (m+2)\epsilon_P - \epsilon_H - m/2^k \end{cases}$$

Here, T denotes the running time of authentication device once.

Performance Measurements of Tor Hidden Services in Low-Bandwidth Access Networks

Jörg Lenhard[1], Karsten Loesing[2], and Guido Wirtz[1]

[1] University of Bamberg, Germany
joerg.lenhard@stud.uni-bamberg.de,
guido.wirtz@uni-bamberg.de
[2] The Tor Project
karsten.loesing@gmx.net

Abstract. Being able to access and provide Internet services anonymously is an important mechanism to ensure freedom of speech in vast parts of the world. Offering location-hidden services on the Internet requires complex redirection protocols to obscure the locations and identities of communication partners. The anonymity system Tor supports such a protocol for providing and accessing TCP-based services anonymously. The complexity of the hidden service protocol results in significantly higher response times which is, however, a crucial barrier to user acceptance. This communication overhead becomes even more evident when using limited access networks like cellular phone networks. We provide comprehensive measurements and statistical analysis of the bootstrapping of client processes and different sub-steps of the Tor hidden service protocol under the influence of limited access networks. Thereby, we are able to identify bottlenecks for low-bandwidth access networks and to suggest improvements regarding these networks.

1 Introduction

With the Internet paving its way into more and more areas of life and business, also the need for privacy on the Internet is ever increasing. The greater the number of users and the wider the area of utilization gets, the greater grows also the number of loopholes that can be exploited through the lack of privacy. But privacy is, among other things, the basis for various core values of democratic societies, like freedom of speech. Hence, providing mechanisms for anonymous communication can be considered an important goal. Privacy is not only relevant for those requesting information or using services offered by others in an anonymous manner. It is also important for providers of services. There is no merit in being able to utter one's opinion freely and without fear of harassment if there is no platform on which one could do so.

There are various approaches to address the subject of anonymous communication. One of them is based on the concept of routing traffic through networks of relays, called *anonymity networks*. The Tor network [6] is a widely deployed anonymity network and consists of approximately 1,300 relays in March 2009. A user of such a network builds a chain of several relays, a *circuit*, to prevent others

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 324–341, 2009.

from linking her identity or location with her actions. All connections between relays are secured using cryptographic mechanisms and none of the relays knows both initiator and responder of a communication session. The user then routes her traffic over the circuit. So, for an outside observer, it looks as if all requests are performed by the last relay in the circuit and not by the actual user. The assumption is that an adversary does not control all relays in a circuit, or more precisely, at least not the first and last relay in a circuit.

Tor permits requesting information from public servers anonymously, as well as providing services pseudonymously, without revealing the IP address of the server. The former functionality is given by attaching application-level streams to circuits which are built as described above. The latter functionality is called *hidden services* and works by connecting circuits of both client and hidden server on a common rendezvous point, again a relay in the network, to grant location privacy to both communication parties. It is obvious that this process is inevitably more complex than connecting to a non-anonymized service.

The usual assumption nowadays is that clients or service providers use broadband access networks of some kind, like cable, DSL, or UMTS. But access networks with lower bandwidth, like second-generation cellular wireless or fixed-line networks are still in wide-spread use. These networks generally provide lower data rates and higher latencies. In many regions of the world, especially less industrialized countries, users are dependent on older, and therefore inferior networks. However, these regions might have an even higher demand for privacy than well-connected areas, as it happens to be the case that they are also less politically stable. The question to be answered here is to what extent the access network of a user influences her capability to use an anonymity network. Studies on the influence of low-bandwidth access networks on the usage of anonymity networks are rare. The present study is the first one to consider the access of location-hidden services using such access networks.

The approach we are taking here is to measure the performance of Tor processes over low-bandwidth access networks, in particular mobile phone and fixed-line telephone networks. We created a measurement setup, involving several Tor processes using these networks, as well as broadband networks. We focus on the evaluation of clients bootstrapping in the low-bandwidth environments and the sub-steps of connecting to hidden services. Both accessing and providing hidden services over low-bandwidth access networks is considered. By these measurements, we identify specific bottlenecks in the process that need to be improved.

In the next section we give a brief overview over previous work on the performance of anonymity networks, especially Tor. Section 3 describes the Tor bootstrapping phase and the Tor hidden service protocol, being the focus of this paper. Section 4 contains an analysis of the proportion of low-bandwidth clients in the Tor network and a description of the environment we created to gather the data. In Section 5 we present statistical analysis of the data from the bootstrapping phase, discuss the implications of this data and suggest performance improvements based on our evaluations. Section 6 contains a similar analysis for hidden services with special focus on circuit building times. Section 7 concludes the paper.

2 Related Work

Work on the performance of anonymity networks is not only motivated by improving usability for its own sake. The level of anonymity provided by the network is dependent on the number of users, forming the underlying *anonymity set*. Networks that offer a high performance will attract more users, resulting in a higher degree of anonymity provided for all participants [5].

Recently, there is a growing interest in measuring performance of anonymous communication. Köpsell [10] observed the influence of the performance provided by a network on the number of users. Wendolsky et al. [18] measured the performance of anonymous communication from the client's point of view. They observed connection latencies to be on average approximately 4 seconds for the Tor network. Utilizing the work of Köpsell, they conclude that these 4 seconds are the overall tolerance level users are willing to take. It is important to note that these 4 seconds cannot be directly compared to this study. Here, we observe accessing and providing hidden services which is necessarily more complex than accessing public services anonymously.

Panchenko et al. [14] focus on the examination of possible reasons for the delay of the Tor network. Their special interest concerns the building of circuits and the geographical diversity of the relays in a circuit. With the help of empirical measurements, they advertise a new path selection algorithm to improve the performance of anonymous communication via Tor.

Øverlier and Syverson [12] suggested changes to the protocol for establishing connections to hidden services. Their suggestions include the reduction of the number of relays involved in the process, which should lead to a decrease in connection establishment times. In earlier studies [11], we measured the latencies during connection establishment to hidden services with special focus on the overall response times. We found that connection establishment, when using a broadband access network, took on average 24 seconds. It is important to mention that these numbers are lower than those presented in this paper. Here, we also consider the time a client needs to build a circuit to a directory server.

However, all studies discussed in this section only consider broadband access networks, neglecting the influence of low-bandwidth access networks as discussed here.

3 Tor Background

Before going into the details of the measurements and their results, some background on the measurement setup is in order. In this section, we describe the Tor bootstrapping process and the Tor hidden service protocol, being the focus of our measurements.

When connecting to the Tor network for the first time, a client needs to download and verify information about the status of the network and single relays in it. [16] describes document formats and [3] outlines the process in detail. As the documents reflect the state of the network, their size can vary

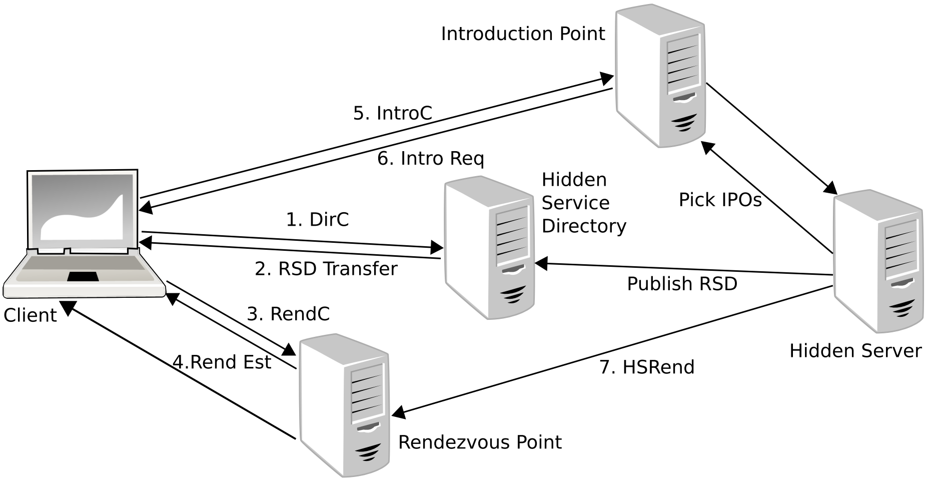


Fig. 1. Establishment and access of a hidden service

strongly, depending on the size of the network. The initial action of a newly started Tor process is to choose a directory authority, establish a TCP connection to it, perform a TLS handshake, and establish a one-hop circuit (bootstrapping phase 0–15%). Next, the Tor process opens a stream to load a network consensus document (15–25%). The process retrieves the document which currently (March 2009) has a size of approximately 90 kilobytes, checks its signatures, and starts loading relay descriptors (25–50%). The process continues loading descriptors until at least one fourth of the total amount is fetched (50–80%). All server descriptors of the network currently add up to approximately 1.6 megabytes of data. Then, the process chooses relays and starts building circuits. For this, again a TCP connection to a relay is built and a TLS handshake is performed. The process then keeps on adding relays to the circuit until it has finished the first circuit consisting of three hops, concluding the bootstrapping process (80–100%). So, all in all about 500 kilobytes of data need to be downloaded by a newly started process to successfully connect to the Tor network. The rest of the data will also have to be downloaded during runtime for ensuring anonymity.

Tor can be used for accessing public services in an anonymous way, but it can also be used to provide services anonymously. The actions described above are independent of hidden services and also apply to normal Tor usage. To be able to communicate with each other anonymously, the provider of the service as well as the client have to perform various steps of the hidden service protocol. Figure 1 visualizes the process of establishing and accessing a hidden service and outlines which steps are measured.

The first step in the hidden service protocol [17] is the establishment of a hidden service in the network by its provider, Bob. For this, Bob configures a Tor process to act as a proxy for his service. The Tor process then builds circuits to three arbitrarily chosen relays in the network and establishes *introduction points* on them for his service. Introduction points work as medium-time contact points for clients trying to access the hidden service. Furthermore, Bob generates a public and a private key for the service and derives a unique identifier from it, the *onion address*. This address consists of sixteen characters ending in *.onion*. As the onion address is derived from the public key, anyone possessing the key can verify that they are communicating with the respective service. In the next step, Bob constructs a *rendezvous service descriptor* (RSD) with the contact information of the introduction points and his public key. He signs the descriptor with his private key and publishes it to a directory server, normally an ordinary relay that provides additional functionality for storing RSDs. Now the hidden service is ready to be accessed by a client, Alice.

First, Alice learns about the hidden service and its onion address and decides to access that service. She needs a Tor process to work as a proxy for her request. She builds a circuit to a directory server (*DirC*) and asks for Bob's RSD. Not all circuits needed during the connection establishment are newly built. If possible, the process tries to pick an existing pre-built circuit. This technique is called *cannibalization* and means that the purpose of a previously built circuit can be changed to whatever purpose is required. This operation can be done without delay. The cannibalized circuit only needs to be extended by a single hop to the directory node. If the RSD is found, Alice downloads it (*RSD Transfer*). She then finds the introduction points' addresses along with Bob's public key in it.

As soon as the RSD is loaded, Alice tries to cannibalize two more circuits. The first one is the circuit to the *rendezvous point* (*RendC*) which is a randomly chosen relay in the network, Alice wants to use for later message exchange with Bob. This circuit has to be a three-hop circuit that can simply be cannibalized, without further operations needed. If no circuit is available for cannibalization, a new three-hop circuit is built from scratch. After completing the circuit to the rendezvous point, Alice establishes it as such (*Rend Est*). This establishment consists of the transmission of two cells. The first cell is sent from Alice to the rendezvous point and requests the rendezvous. The second cell is sent from the rendezvous point to Alice and acknowledges the request. When requesting the rendezvous, Alice also hands over a one-time secret, serving as her identification. The second circuit built after the reception of the RSD is a circuit to one of the introduction points of the hidden service (*IntroC*). As soon as a circuit to an introduction point is completed and a rendezvous point is established, Alice requests this relay to introduce herself to the service (*Intro Req*). She does this by handing over the rendezvous point's address and her one-time secret, encrypted with Bob's public key. The introduction point answers with an acknowledgment message and forwards Alice's request and her secret to the hidden service. Bob decrypts this message using his private key and obtains the address of the rendezvous point and Alice's one-time secret. Bob can now decide if he wants to contact Alice, and if so, he builds a circuit to the rendezvous point (*HSRend*). When this circuit is completed, Bob asks the rendezvous point to connect his circuit to Alice's. The rendezvous point matches their circuits with the help of the one-time secret and establishes a connection between the two. Then, the rendezvous point sends Alice a notification about the connection establishment.

Now, Alice and Bob can start exchanging messages via the rendezvous point. We denote the period from request start to reception of the connection establishment message, sent from the rendezvous point to Alice, as total round-trip time (*Total RTT*) and the period from reception of the RSD to reception of the same connection establishment message as small round-trip time (*Small RTT*).

4 Measurement Setup

The measurement setup consists of a few Tor processes connected to the public Tor network over either broadband or low-bandwidth access networks. In this section, we give some information about the low-bandwidth access networks and describe the utilized Tor versions and process distribution.

The access networks we observed are analog modulation via the telephone network, the mobile network *Enhanced Data Rates for GSM Evolution* (EDGE) [15], and a broadband network. The modem we used was of standard *V92*, thus offering a data rate of 56 kilobits per second downstream and 44 kilobits per second upstream [9]. EDGE provides a data rate of up to 230 kilobits per second, depending on radio conditions. The broadband connection was represented by the university network, consisting of fiber optics and offering a data rate of up to 100 megabits per second. For the Tor processes a minimal fraction of this rate would have been sufficient, so the broadband access network can be considered unlimited for the measurements.

These access networks can be seen as representatives of the prevailing types of access networks nowadays. Analog modulation formed the most important access network in the early times of the Internet. Although it is losing its importance more and more, it is still widely in use, especially in developing countries and rural areas where the establishment of broadband networks is not yet lucrative for ISPs. Moreover, many desktop and laptop computers are also equipped with V92 modems per default. EDGE is an enhancement of the GSM standard for mobile communication which was established in 1982. As of September 2008, GSM makes up eighty percent of the world's subscriber connections [7]. In contrast to broadband mobile access networks, such as UMTS, EDGE and its predecessor enhancement of GSM, GPRS [15], are widely distributed in industrialized countries and also available in less developed areas. Data transmission via optical fibers provides the highest possible data rates today. It is not yet forming a major access network, due to its expense. Instead, it is generally combined with other fixed line access networks. Fiber optics connects main centers, whereas the final connections to households are built using, for example, DSL.

When conducting the measurements, we assumed that low-bandwidth access networks are used by a major share of the networks' clients. Based on a suggestion from one of the reviewers, we investigated this assumption more closely. We observed the bandwidth of clients downloading the network consensus document from one of the six directory authorities for one week between March 14–21, 2009. We analyzed the size and duration of every consensus document download to conclude which bandwidth clients have. We excluded relays to observe only

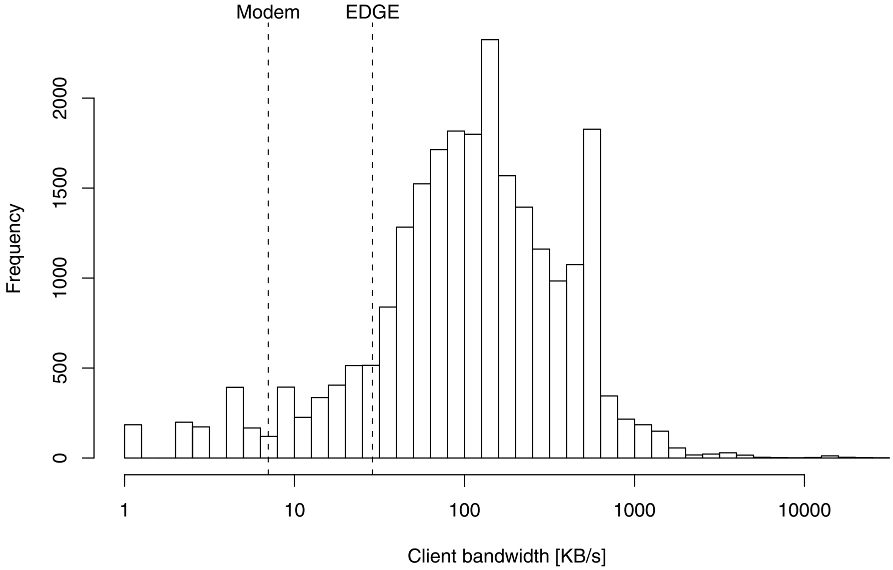


Fig. 2. Download speed of client connections loading the network consensus in log scale

the bandwidth of clients. Results are shown in Figure 2. Roughly 7% of these connections might have been performed by clients using a V92 modem, and 16% by clients using EDGE or a modem. So, a total of 16% of the network's clients can be considered low-bandwidth in the terms of this study. Our measurements only include successful downloads, so that the number of low-bandwidth clients might be even higher. This is a sufficiently large share to demand special interest. Furthermore, if the network had lower bandwidth requirements, the number of low-bandwidth clients might increase even more.

We observed the log events of the Tor processes indicating the sending and reception of messages, the opening of circuits and the progress in the bootstrapping phase. Client and hidden service processes used Tor version 0.2.1.5-alpha as code base. We had to patch this version to resolve two bugs that would otherwise have had an impact on the measurements.[1]. The first bug involved failures when loading router descriptors, and the second bug lead to erroneous behavior when loading rendezvous service descriptors. Both bugfixes are contained in Tor version 0.2.1.6-alpha which was not available at the time of performing measurements. So, we patched the Tor versions of clients and hidden services with all revisions that were necessary to fix these bugs.[2]

We further implemented a few changes to the Tor source code in order to perform measurements: The first change forces the Tor process offering a hidden

[1] Detailed descriptions can be found in Tor's bug tracker: `http://bugs.noreply.org/flyspray/index.php?do=details&id=767` and `http://bugs.noreply.org/flyspray/index.php?do=details&id=814`

[2] These were the revisions r16808, r16810, r16817, and r16915.

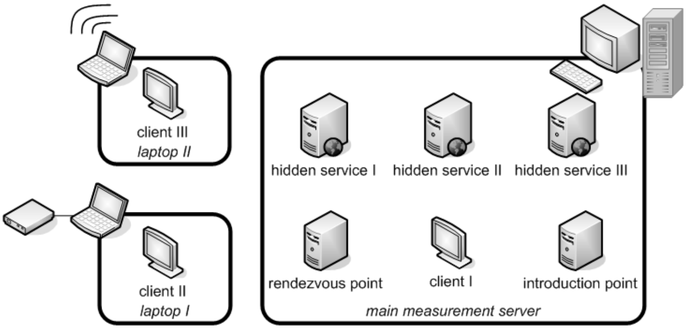


Fig. 3. Process distribution with clients using low-bandwidth access networks

service to select a specific relay as introduction point which can be controlled by us. The second change is to make clients pre-build a three-hop circuit to a specific relay which is also controlled by us, so that it can be chosen as rendezvous point later on. As a third source code change, the client selects the introduction point that is controlled by us, given that Bob had chosen it in the first place. For the measurements we set up a Tor relay, acting as introduction and rendezvous point. This Tor process was running an unpatched Tor 0.2.1.4-alpha version, as neither introduction nor rendezvous point were affected by the previously mentioned bugs. The measurements were then divided into two phases. During all measurements, three hidden services, one for each access network type, and the relay were running continuously.

The Tor processes for the hidden services as well as the introduction and rendezvous point were started some time prior to the measurements. Clients accessing the services were created in regular intervals. We did not use the same Tor processes for the clients, but created new ones in each interval, to avoid any influences on the results by caching directory information. The distribution of the processes on the different physical machines is shown in Figures 3 and 4, respectively. In the first measurement phase, the clients used low-bandwidth access networks, while the hidden services had broadband access. In the next measurements the configuration was turned around and the clients used the broadband access network, while the hidden services were offered over low-bandwidth access networks. Every access network was used by a different physical machine and the low-bandwidth access networks were only used by one Tor client or hidden service at a time to not overcharge them. So, all in all, we used three computers, the main measurement server and two laptops. All other processes were running on the machine using the broadband access network. However, this was not a problem, because all processes communicated over circuits in the real Tor network and never directly.

The interval in which client processes were created, and thus the time they had to bootstrap and perform the hidden service request, was capped at 6 minutes for the client-side low-bandwidth measurements and 5 minutes for the server-side low-bandwidth measurements. During both measurements, as soon as the client process finished bootstrapping, the hidden service request was initiated.

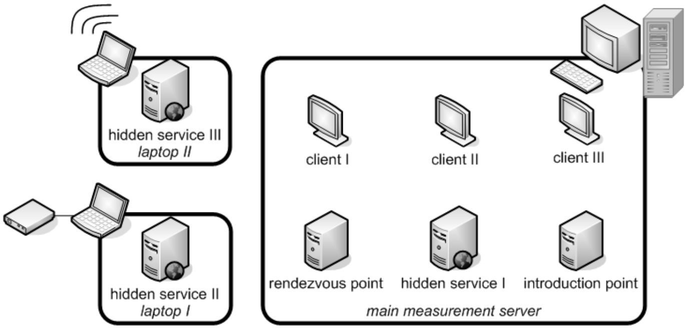


Fig. 4. Process distribution with hidden services using low-bandwidth access networks

We chose different intervals for both measurements due to the bootstrapping phase. During the server-side low-bandwidth measurements, hidden service processes were running over low-bandwidth networks. These processes needed to bootstrap only once in advance to the measurement period. During the other measurements, bootstrapping by the client processes needed to take place in every interval, also consuming more time. Client-side low-bandwidth measurements then lasted for 134 hours between 23–29 September 2008 and server-side low-bandwidth measurements for 114 hours between 6–11 October 2008.

5 Bootstrapping

As a first step in analysis, we investigate the total bootstrapping time as visualized in Figure 5. It becomes clear that bootstrapping over limited access networks is a major problem in comparison to the broadband access network. For the limited networks, the total bootstrapping time is approximately five times that of the broadband network, with median values of 232.9 seconds for EDGE and 249.0 seconds for modem and an interquartile range of 91.9 seconds for EDGE and 45.6 seconds for modem. The broadband median lies at 22.9 seconds and the interquartile range at 39.3 seconds. It is important to note that descriptive values are likely to be even higher in the population, especially for maximum values. In the measurements, test runs were stopped after 6 minutes which eliminated records that would have exceeded this value.

It has turned out that some sub-steps of the bootstrapping process contribute more to these differences than others, as can be seen in Figure 6. It is obvious that the most time-consuming sub-step lies between fifty and eighty percent, where relay descriptors are loaded. At the time of measurement, at least 325 relay descriptors had to be loaded during this phase to initiate the building of circuits. These descriptors make up the largest share of the data that needs to be downloaded during bootstrapping. The median duration of this sub-step ranges from 127.0 seconds for EDGE to 155.6 seconds for modem which is more than forty times as long as the broadband network with 3.3 seconds. This is a

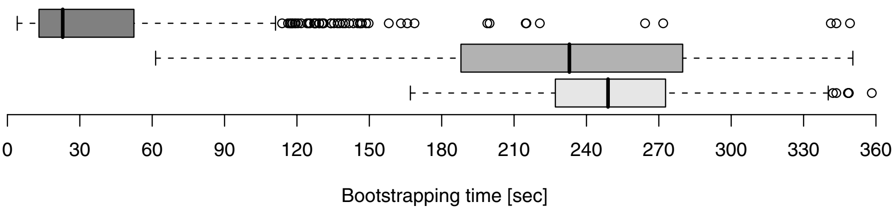


Fig. 5. Durations of total bootstrapping time [sec] in broadband (dark gray), EDGE (medium gray), and modem (light gray) access networks

considerable barrier for the usage of the anonymity network over limited access networks.

A reduction of the initial amount of descriptors that need to be downloaded is not an option, as it would pose a threat to anonymity which is of course more important than performance. The smaller the initial set of relays a client is able to choose from, the more the client is prone to *intersection attacks* [2]. For these attacks, it is necessary that the users of the network are not continuously active and some messages might be linkable. If an attacker knows the initial set of relays a client might use, she could observe messages sent via these relays at a given point of time and intersect the sets of possible active senders, thus cutting out the non-active users at this point of time and reducing the sets of possible senders. The smaller the initial set of relays, the easier this operation gets. By systematically reducing the sets of possible senders, an attacker could correlate messages to certain clients.

The problem of slow bootstrapping is also addressed by several Tor proposals. One approach is to drop the requirement to download server descriptors while bootstrapping [13] and download them on demand while building circuits. In this approach, clients would still be able to use all relays for circuit building. The idea is to add all information that is required for path selection into the network summary, so that server descriptors are only required for building circuits. This approach reduces the download size of directory information during bootstrapping from at least 500 kilobytes to 100 kilobytes. The disadvantage of the described approach is, however, that all circuit extensions require an additional message to download the required server descriptor. Clients must not cache received server descriptors for future extensions, because this would leak the information that a client has used a relay before from not having to ask for its descriptor. As a result, the improvement in bootstrapping leads to deterioration in circuit establishment. A subsequent proposal [4] introduces microdescriptors containing only the onion key as the minimum information for building circuits. Clients would download microdescriptors instead of router descriptors, reducing the total size of directory information during bootstrapping to around 300 kilobytes. It is yet uncertain which variant will be implemented in future Tor versions. But the discussion shows that there is a need to find better solutions to accelerate the bootstrapping process and support clients on low-bandwidth access networks.

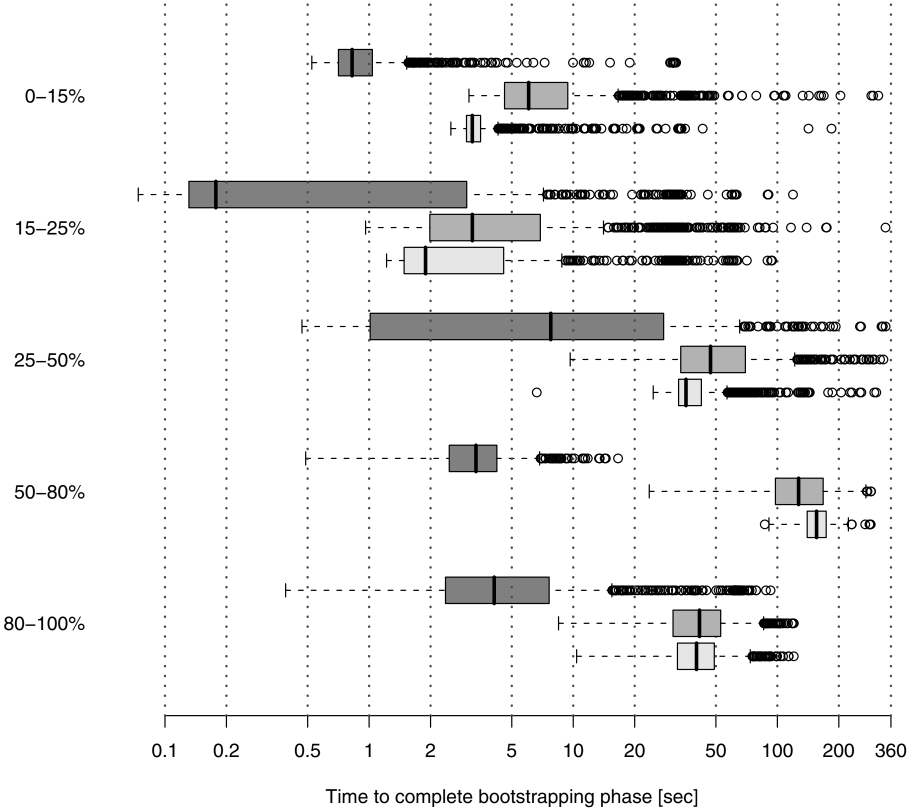


Fig. 6. Durations of bootstrapping substeps [sec] in broadband (dark gray), EDGE (medium gray), and modem (light gray) access networks on a logarithmic scale

6 Hidden Service Access

The second focus of this paper lies on hidden service access times. It has to be stated that bootstrapping took far longer than we had expected. This had an effect on the measurements of hidden service connection establishment. If the bootstrapping phase took up most of the whole measurement interval, there was no time to perform the actual hidden service request. This lack of time resulted in a cut-off and missing values at some point during the process. For the evaluation of the hidden service requests, we limited the data set to those requests that were not influenced by the bootstrapping phase. That is to say, only requests are considered that had at least 2 minutes of the measurement interval left. However, these restrictions only affect the data of the client-side low-bandwidth measurements, as only here bootstrapping was a major issue. Instead of 1,350 hidden service requests performed, we limited the data set of the low-bandwidth access networks to around 500 records. For these records, independence from the bootstrapping phase is guaranteed.

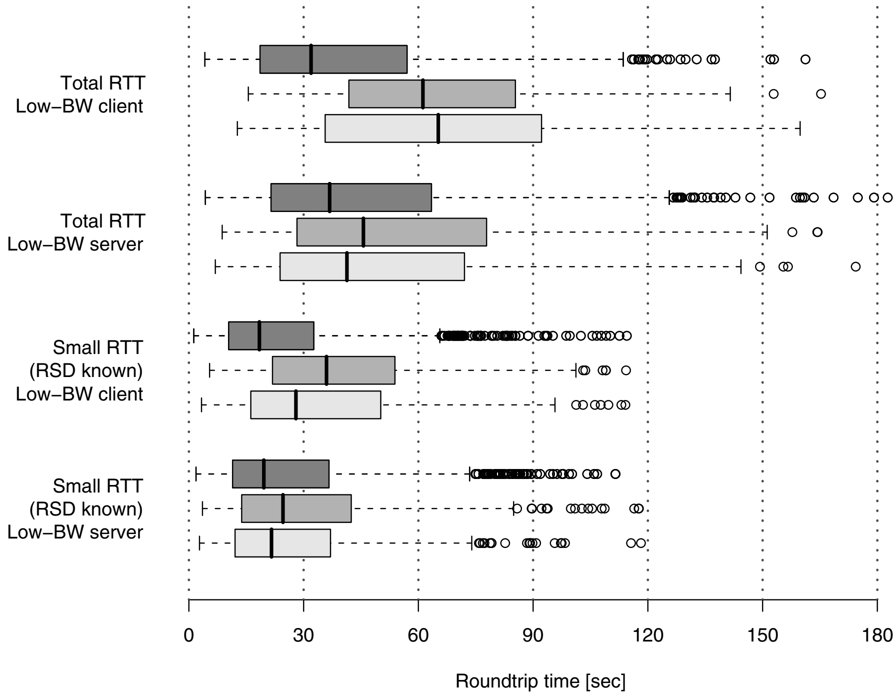


Fig. 7. Durations of round-trip times [sec] in broadband (dark gray), EDGE (medium gray), and modem (light gray) access networks

Figure 7 outlines round-trip times for clients and servers on low-bandwidth access networks. For the client-side measurements, the differences between the access networks are obvious. When considering the total RTT, median values range at up to more than twice as high for the limited access networks, with a total RTT of 61.2 seconds for EDGE and 65.2 seconds for modem in comparison to 32.0 seconds for broadband. The interquartile range lies at 43.5 seconds for EDGE, 56.6 seconds for modem and 38.4 seconds for broadband. For the small RTT, the differences between broadband and limited networks shrinks to 17.6 seconds for EDGE and 9.6 seconds for modem, when comparing the median. Absolute median values and interquartile range amount to 36.0 seconds and 31.7 seconds for EDGE, 28.0 seconds and 34 seconds for modem as well as 18.4 seconds and 22.2 seconds for broadband. For the server-side low-bandwidth measurements, the difference is less obvious. When looking at the total RTT median, it shrinks to 8 seconds between EDGE and broadband and only 1 second between modem and broadband, with absolute values of 44.2 seconds for EDGE, 37.7 seconds for modem and 36.4 seconds for broadband. The small RTT shows similar results with a difference of 8.6 and 2.5 seconds, respectively, when comparing the broadband network to EDGE, respectively modem. Absolute median values range at 25.8 seconds for EDGE, 19.7 seconds for modem and 17.2 seconds for broadband.

When looking at the round-trip times in the server-side low-bandwidth measurements, we can observe that values for the low-bandwidth access networks do not differ strongly from those of the broadband network. Especially the values of the modem network range at a level of only approximately 1 second higher. An analysis of the different sub-steps of the whole protocol unveils the reasons for this. For events where the hidden service access network is involved, broadband shows a better performance. But these events have a much smaller impact on the total access time than events dependent on the client access network. For the client-side events, we observed a slightly better performance of clients accessing services with a low-bandwidth access network, ranging at a level of 0.1 seconds per event. These discrepancies have to be assumed to be random, because all respective processes were running on the same physical machine using the same access network. There is no way in which client processes accessing low-bandwidth services could have been preferred over other processes. Still, these random differences equalize the differences produced by the hidden service access network. This becomes especially obvious for the modem connection where, in the end, there is hardly any difference to the broadband connection. We conclude that the influence of the hidden service access network on the hidden service protocol is of rather minor importance. Hidden services can in principle be offered over low-bandwidth access networks, without enlarging the overall connection establishment time too much. Other factors might be more likely to produce a bottleneck here. These could, for example, be the usage of the access network for something besides offering the hidden service, thus limiting the available bandwidth even further. Also the size of the actual product of the service or the number of clients accessing the service at the same time are relevant.

We concentrated our further analysis of hidden service access on circuit establishment. The building of the various circuits consumes the largest share of time in the whole process, in many cases up to 80% of the total access time. Once circuits are completed, message transfer times only constitute minor delays. Figure 8 shows establishment times for all circuits involved in the process of accessing a hidden service. For the completion of each of the circuits there is a timeout of 60 seconds. If the circuit is not completed within this time, it is abandoned and a new attempt is started. It is important to mention that the presented data constitutes absolute times until a circuit to a respective relay is established. This can involve more than one attempt and thus also more than 60 seconds.

The *client-side circuit to the rendezvous point (RendC)* is built very quickly for all access networks, almost immediately after requesting it. The median values are 0.0 seconds for all access network types. This is the case, because the rendezvous circuit is simply cannibalized and not extended. In very few cases, cannibalization was not possible, and a new circuit had to be built which of course took some more time.

The *client-side circuits to the directory server (DirC) and introduction point (IntroC)* show bigger differences between the access network types. Values for these circuits are very similar for the same network types, as they are built

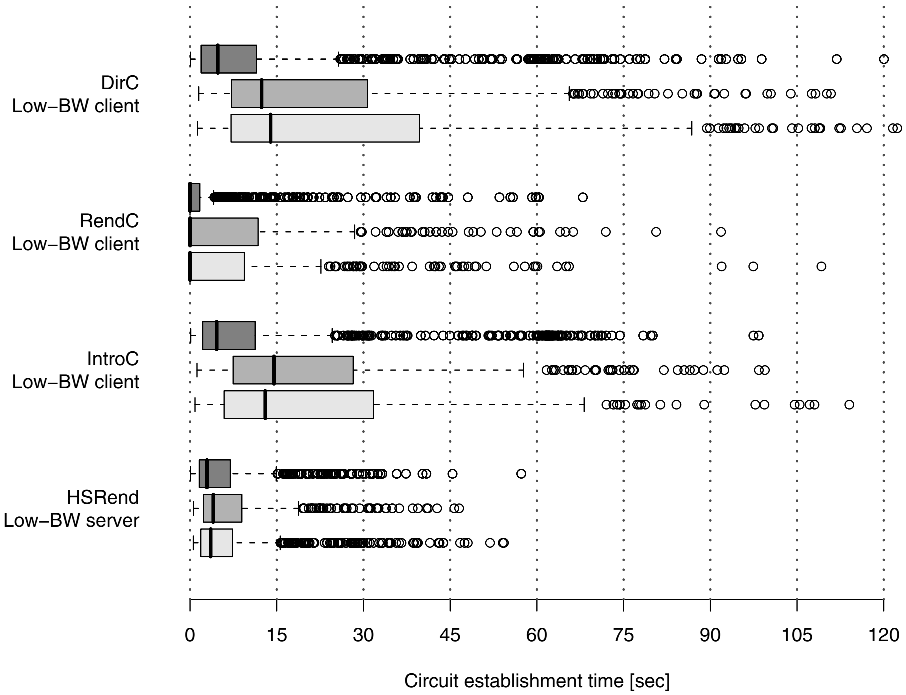


Fig. 8. Durations of circuit building [sec] in broadband (dark gray), EDGE (medium gray), and modem (light gray) access networks

in the same manner. Here, if possible, an existing circuit is cannibalized and extended to the respective relay. This extension has noticeable impact for the different access networks. This can be seen by the high difference, compared to the broadband network, in median values. In median, for both circuits, values range about 8 to 9 seconds higher for the limited access networks. It has to be mentioned that the data for the circuits to the directory server presented here is likely to be slightly higher than in the population. In some cases the time of the completion of this circuit could not be determined unambiguously from the log files among other circuits. We considered a slight over-estimation to be less critical and thus always chose the circuit that finished last.

Finally, the *hidden-service-side circuit to the rendezvous point (HSRend)* is built rather quickly and the broadband network is only slightly faster with around 1 second in difference for median in comparison to the low-bandwidth access networks. This circuit is cannibalized and extended to the rendezvous point. The hidden services in the measurements had fewer operations to perform than the clients. Bootstrapping was done once and fewer circuits had to be built during an attempt. So, the hidden services were more likely to have an existing internal circuit ready for cannibalization which explains why the building time for this circuit is much shorter than the time for building the client circuit to the introduction point or to the directory server.

Table 1. Binomial tests on circuit completion

Phase	Type	p_{30}	p_{40}	p_{45}
DirC	Broadband	$4.1e^{-8}$	$6.9e^{-17}$	$4.5e^{-19}$
	EDGE	1	0.04	$9.2e^{-4}$
	Modem	1	0.52	0.06
IntroC	Broadband	$1.1e^{-4}$	$9.4e^{-16}$	$2.8e^{-21}$
	EDGE	1	0.26	$1.2e^{-3}$
	Modem	1	0.58	0.03
HSRend	Broadband	$2.1e^{-14}$	$7.2e^{-25}$	$1.6e^{-27}$
	EDGE	$2.0e^{-12}$	$6.4e^{-24}$	$1.5e^{-26}$
	Modem	$2.1e^{-14}$	$3.0e^{-27}$	$1.8e^{-33}$

Starting with Tor version 0.2.1.7-alpha, the circuit timeout for the above circuits has been reduced to 30 seconds and in case of the introduction circuit, after 15 seconds a second attempt is started in parallel. It can now be observed with the present data if this new timeout is suitable also when limited access networks are in use. To determine the suitability, we applied binomial tests [8]. This type of test simply requires a binomial distribution of the data set. So, we split the set into two groups: on the one hand those attempts where the building of a certain circuit took less or equal time than for example 30 seconds and on the other hand those where it took more, up to 60 seconds. As the timeout is only relevant for a single attempt, we did not consider the absolute times as represented in Figure 8, but instead analyzed single attempts. We considered all successful attempts, no matter whether they were the first or second or maybe even third try to build a circuit to a certain relay. We set the percentage of completed circuits for considering a timeout as suitable, to 90%. Put in other words, concerning the binomial tests, we set the probability for a success to 0.9. It was important to find a measure for the timeout that guaranteed fault recognition, without cutting off too many attempts that would have finished later. Furthermore, Panchenko et al. showed that subsequent message transmission times over a circuit correlate to its building time [14]. So, cutting off circuit building at a reasonable limit should increase the performance of connection establishment and message transmission. We set the significance level to 5%. As the tests were three-fold, because of three connection types, we applied alpha adjustment which reduced the significance level to 1.66%. We did not perform binomial tests for the client circuit to the rendezvous point. This circuit is built almost immediately in most cases and a timeout reduction would not advance this. The results of the tests can be found in Table 1. It is quite obvious that a timeout of 60 seconds is too high in case of the hidden service circuit to the rendezvous point. Very low and significant p-values are achieved for all access network types. Thus, a timeout reduction to 30 seconds for this circuit is reasonable. But for the other circuits, the timeout reduction cannot be supported with the present data. While the broadband access network shows significant p-values also for 30 seconds, the low-bandwidth networks do not. Even a timeout of 40 seconds does not fit. The p-value of EDGE for the circuit to the directory server with 0.4 approaches a significant level, but other p-values still rank high. Only with a timeout of 45 seconds, p-values of EDGE become significant. The p-values of the modem access network, with 0.06 for the circuit to the directory server and 0.03 for the circuit to the introduction point, are still not significant but close to the significance level. As the timeout should be as convenient as possible for all access network types, a compromise needs to be chosen. On the one hand, a timeout of 45 seconds might still be slightly too low for the modem access network. On the other hand it is provably too high for the broadband access network. Being the convenient middle way, we propose a timeout of 45 seconds for both circuits.

Improving static timeouts may be a good first step. But as our measurements show, no timeout can fit all client environments equally well. A better approach would be to track circuit build times at the client and use these data to adjust a local timeout variable. By doing so, clients could even adapt to changing network environments. One such approach is described in a Tor proposal [1] which is not yet implemented, though.

7 Conclusion

We conducted comprehensive performance measurements of the usage of Tor in limited access networks. Our focus was the evaluation of the bootstrapping phase and sub-steps of hidden service access, especially circuit building and round-trip times. The bootstrapping phase has turned out to take significantly longer than expected over low-bandwidth access networks. The bottleneck in this phase is formed by the download of relay descriptors. We discussed advantages and disadvantages of different approaches to accelerate the bootstrapping process. The analysis of circuit building times showed that building or extending circuits is a major bottleneck in the process of accessing hidden services, especially when using low-bandwidth access networks. We conducted binomial tests to determine adequate timeouts for the circuits involved in hidden service access. We confirmed the usefulness of the timeout for the hidden service circuit to the rendezvous point. For the client circuit to the directory server and to the introduction point, we showed that the timeout is set too small when using low-bandwidth access networks. Instead we proposed a timeout of 45 seconds for these two circuits which would also fit the demands of the limited access networks. Furthermore, we found round-trip times to not differ strongly when using service-side low-bandwidth access networks. For the usage of client-side low-bandwidth access networks, the difference was more obvious.

The contribution of this paper is to compare a few selected uses of anonymity networks in low-bandwidth access networks. Future investigations might focus on other use cases, e.g., anonymous web surfing or downloading of large files. Also, other types of anonymity networks could be taken into consideration. Further future work includes separate measurements of bootstrapping and Tor hidden services in low-bandwidth environments. The limitation of the measurements to 6 minutes reduced the data that could be collected. With significantly shorter bootstrapping, the 6 minutes could be used to measure application-level message latency or throughput. For measurements of the bootstrapping, a higher timeout of 10 to 15 minutes could give more informative results.

Acknowledgements

We thank the reviewers for their helpful comments which gave us the idea to measure and include data on client bandwidths in the Tor network. Part of this work has been generously funded by the NLnet foundation.

References

1. Chen, F., Perry, M.: Improving Tor path selection. Tor Proposal 151, The Tor Project (July 2008), `https://svn.torproject.org/svn/tor/trunk/doc/spec/proposals/151-path-selection-improvements.txt`
2. Danezis, G., Serjantov, A.: Statistical disclosure or intersection attacks on anonymity systems. In: Fridrich, J. (ed.) IH 2004. LNCS, vol. 3200, pp. 293–308. Springer, Heidelberg (2004)
3. Dingledine, R.: Keep controllers informed as Tor bootstraps. Tor Proposal 137, The Tor Project (July 2008), `https://svn.torproject.org/svn/tor/trunk/doc/spec/proposals/137-bootstrap-phases.txt`
4. Dingledine, R.: Clients download consensus + microdescriptors. Tor Proposal 158, The Tor Project (January 2009), `https://svn.torproject.org/svn/tor/trunk/doc/spec/proposals/158-microdescriptors.txt`
5. Dingledine, R., Mathewson, N.: Anonymity loves company: Usability and the network effect. In: Anderson, R. (ed.) Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006), Cambridge, UK (June 2006)
6. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th USENIX Security Symposium (August 2004)
7. GSM Assocication. Market Data Summary (2008), `http://www.gsmworld.com/newsroom/market-data/market_data_summary.htm`
8. Hays, W.L.: Statistics. In: Holt, Rinehart, Winston (eds.), 3rd edn. (1981) ISBN: 0-03-056706-8
9. International Telecommunications Union. V.92: Enhancements to Recommendation V.90 (November 2000), `http://www.itu.int/rec/T-REC-V.92-200011-I/en`
10. Köpsell, S.: Low latency anonymous communication – how long are users willing to wait? In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 221–237. Springer, Heidelberg (2006)
11. Loesing, K., Sandmann, W., Wilms, C., Wirtz, G.: Performance Measurements and Statistics of Tor Hidden Services. In: Proceedings of the 2008 International Symposium on Applications and the Internet (SAINT), Turku, Finland. IEEE CS Press, Los Alamitos (2008)
12. Øverlier, L., Syverson, P.: Improving efficiency and simplicity of Tor circuit establishment and hidden services. In: Borisov, N., Golle, P. (eds.) PET 2007. LNCS, vol. 4776, pp. 134–152. Springer, Heidelberg (2007)
13. Palfrader, P.: Download server descriptors on demand. Tor Proposal 141, The Tor Project (June 2008), `https://svn.torproject.org/svn/tor/trunk/doc/spec/proposals/141-jit-sd-downloads.txt`

14. Panchenko, A., Pimenidis, L., Renner, J.: Performance analysis of anonymous communication channels provided by Tor. In: ARES 2008: Proceedings of the 2008 Third International Conference on Availability, Reliability and Security, Washington, DC, USA, pp. 221–228. IEEE Computer Society Press, Los Alamitos (2008)
15. Sauter, M.: Communication Systems for the Mobile Information Society. Wiley, Chichester (2006)
16. The Tor Project. Tor directory protocol, version 3 (2008), `https://svn.torproject.org/svn/tor/trunk/doc/spec/dir-spec.txt`
17. The Tor Project. Tor Rendezvous Specification (2008), `https://svn.torproject.org/svn/tor/trunk/doc/spec/rend-spec.txt`
18. Wendolsky, R., Herrmann, D., Federrath, H.: Performance comparison of low-latency anonymisation services from a user perspective. In: Borisov, N., Golle, P. (eds.) PET 2007. LNCS, vol. 4776, pp. 233–253. Springer, Heidelberg (2007)

Cryptanalysis of Twister

Florian Mendel, Christian Rechberger, and Martin Schläffer

Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
martin.schlaeffer@iaik.tugraz.at

Abstract. In this paper, we present a semi-free-start collision attack on the compression function for all Twister variants with negligible complexity. We show how this compression function attack can be extended to construct collisions for Twister-512 slightly faster than brute force search. Furthermore, we present a second-preimage and preimage attack for Twister-512 with complexity of about 2^{384} and 2^{456} compression function evaluations, respectively.

Keywords: SHA-3, Twister, hash function, collision-, second-preimage-, preimage attack.

1 Introduction

In the NIST SHA-3 competition, many new hash function designs have been submitted. NIST published a list of 51 first round candidates, and Twister [4,5] is one of them. The next step is to reduce the list of 51 candidates to a small set of finalists within the next few years. As an input to this selection process, we describe several cryptanalytic results on Twister in this paper. Similar results on the Whirlpool [1], Grøstl [15] and GOST [14] hash functions, or the Merkle-Damgård [3,11] constrution have been published in [10], [8,9] and [6]. Our results on Twister are summarized in Table 1.

Table 1. Summary of cryptanalytic results on Twister

type of attack	target	hash size	complexity	memory
semi-free-start collision	compression function	all	2^8	-
collision	hash function	512	2^{252}	2^9
second preimage	hash function	512	2^{384+s}	$2^{10} + 2^{64-s}$
preimage	hash function	512	2^{456}	2^{10}

In the remainder of the paper, we first give a description of Twister in Section 2, outline a practical collision attack on its compression function in Section 3, and give theoretical collision-, second preimage-, and preimage attacks in Sections 4, 5, and 6, respectively. We summarize and conclude in Section 7.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 342–353, 2009.

2 Description of Twister

The hash function Twister is an iterated hash function based on the Merkle-Damgård design principle. It processes message blocks of 512 bits and produces a hash value of 224, 256, 384, or 512 bits. If the message length is not a multiple of 512, an unambiguous padding method is applied. For the description of the padding method we refer to [4] and [5]. Let $m = m_1 \| m_2 \| \cdots \| m_t$ be a t-block message (after padding). The hash value $h = H(m)$ is computed as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, m_i) \quad \text{for } 0 < i \leq t \\ H_{t+1} &= f(H_t, C) \\ h &= \Omega(H_{t+1}) \ , \end{aligned}$$

where IV is a predefined initial value, C is the value of the checksum and Ω is an output transformation. The checksum C is computed from the intermediate values of the internal state after each Mini-Round. Note that while for Twister-224/256 the checksum is optional it is mandatory for Twister-384/512.

The compression function f of Twister basically consists of 3 Maxi-Rounds. Each Maxi-Rounds consist of 3 or 4 Mini-Rounds (depending on the output size of Twister) and is followed by a feed-forward XOR-operation.

The Mini-Round of Twister is very similar to one round of the Advanced Encryption Standard (AES) [13]. It updates an 8×8 state S of 64 bytes as follows:

MessageInjection. A 8-byte message block M is inserted (via XOR) into the last row of the 8×8 state S.

AddTwistCounter. A 8-byte block counter is xored to the second column of the state S.

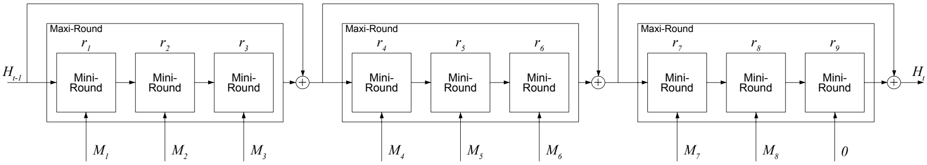


Fig. 1. The compression function of Twister-224/256

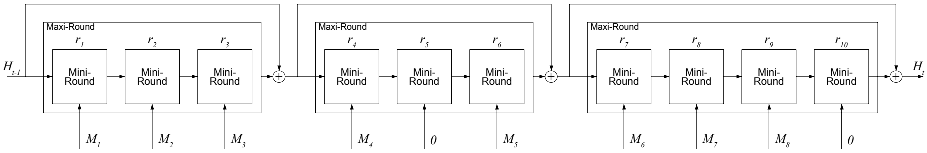


Fig. 2. The compression function of Twister-384/512

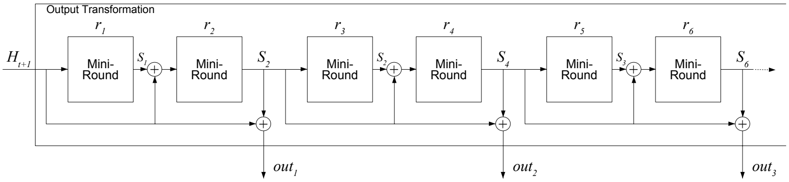


Fig. 3. The output transformation of Twister

SubBytes. is identical to the SubBytes operation of AES. It applies an S-Box to each byte of the state independently

ShiftRows. is a cyclic left shift similar to the ShiftRows operation of AES. It rotates row j by $(j-1) \pmod 8$ bytes to the left.

MixColumns. is similar to the MixColumns operation of AES. It applies a 8×8-MDS matrix A to each column of the state S. The matrix A and its inverse B are given in Appendix A.

After the last message block and/or the checksum has been processed, the final hash value is generated from the last chaining value H_{t+1} by an output transformation Ω (see Figure 3).

In the output transformation, two Mini-Rounds are applied to subsequently output 64 bits of the hash value. The output stream consists of the XOR of the first column of the state prior and after the two Mini-Rounds. This 64-bit output stream continues until the full hash size has been received. For a more detailed description of Twister we refer to [4] and [5].

3 Semi-Free-Start Collision for the Compression Function

In this section, we present a semi-free-start collision attack on the compression function of Twister for all output sizes. The complexity to find a differential characteristic is about 2^8 compression function evaluations. However, for each differential characteristic, we can construct up to 2^{64} message pairs and the complexity to find one of these conforming message pairs is *one*.

In the attack we use the differential characteristic of Figure 4 for the first Maxi-Round (3 Mini-Rounds) of Twister. The 3 Mini-Rounds are denoted by r_1, r_2 and r_3 and the state after the Mini-Round r_i is denoted by S_i and the state after the corresponding feed-forward S_i^F. The initial state or chaining value is denoted by S_0. In the attack we add a difference in message word M_1 (8 active bytes) to the state S_0, which results in a full active state S_1 after the first Mini-Round r_1. After the MixColumns transformation of the second Mini-Round r_2, the differences result in 8 active bytes of the last row of state S_2, which can be canceled by the message word M_3 in the third Mini-Round r_3.

The message differences and values for the state are found using a *rebound* approach as proposed in [10]. Figure 5 shows the characteristic in detail. We start with message word differences in M_1 and M_3 at states S_1' and S_2 (we do not use differences in M_2 to simplify the description of the attack). The differences can

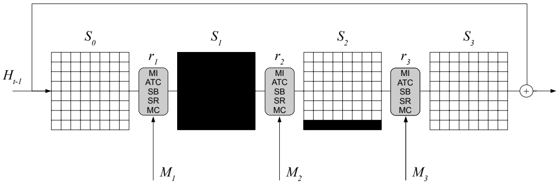


Fig. 4. Characteristic to construct a semi-free-start collision in the first Maxi-Round

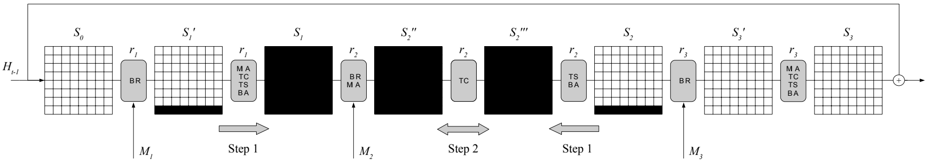


Fig. 5. We start with differences in states S_1' and S_2 injected by message words M_1 and M_3, and propagate backward and forward (Step 1) to find a match for the S-box of round r_2 (Step 2)

be propagated backward and forward through the MixColumns transformation with a probability of one (Step 1). Then, we simply need to find a match for the resulting input and output differences of the SubBytes layer of round r_2 (Step 2) and propagate outwards.

Step 1. We start the attack with 8 active bytes in state S_1''' and S_2 (injected by message words M_1 and M_3) and compute backward and forward to two full active states S_2'' and S_2'''. This happens with a probability of one due to the properties of the ShiftRows and MixColumns transformations. Note that we can significantly reduce the complexity of the attack, if we first compute the full active state S_2''' and then compute S_2'' column by column to find a match for 8 S-boxes at once (Step 2).

Step 2. Next, we show how to find a match for the input/output differences of the 64 active S-boxes of round r_2. Note that for a single S-box, the probability that a input/output differential exists is about one half, and for each *valid* input/output differential we can assign at least two possible values to the S-box (for more details we refer to [10]). Note that we can search for valid S-box differentials for each column independently. Hence, we start by choosing a random difference for the first active byte of S_1''' and compute the corresponding row of S_2''. We find a valid differential match for these 8 S-boxes with a probability of $(1/2)^8 = 2^{-8}$. If we find a match, we continue with the remaining active bytes. Alltogether, this step has a complexity of less than 2^8 compression function evaluations.

Once we have found a differential match for the SubBytes layer, we can choose from at least 2^{64} possible states for S_2''. Each of these states can be computed forward and backward and results in a semi-free-start collision for one

Maxi-Round. Further, this determines the state S_0 as well as the values and differences of M_1 and M_3 (we can freely choose the values of M_2). Note that the first Maxi-Round is the same for Twister-224/256 and Twister-384/512. Hence, by constructing a semi-free-start collision for the first Maxi-Round we already get a semi-free-start collision for the compression function of Twister-224/256 and Twister-384/512. Since we can find 2^8 semi-free-start collisions with a complexity of 2^8 compression function evaluations, the average complexity to find one semi-free-start collisions is one with negligible memory requirements. An example for a semi-free start collision is given in 2.

Table 2. A colliding message pair (M, M^*) for the semi-free-start collision of the first Maxi-Round (S_3, S_3^*) of Twister. The corresponding semi-free-start collision for Twister-256 (with output transformation) is given by $H(256)$ and $H^*(256)$.

H_0	A63215B04567E389	16D40B5ACFABED9D	C0C4104853084862	C38990B8BEBF7BED
	E936F9AF6406E35B	F5BE6C8455626226	C6C9FA7B806B3BD1	E22C576CDE8ABDB5
H_0^*	A63215B04567E389	16D40B5ACFABED9D	C0C4104853084862	C38990B8BEBF7BED
	E936F9AF6406E35B	F5BE6C8455626226	C6C9FA7B806B3BD1	E22C576CDE8ABDB5
ΔH_0	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000
M	0000000000000000	0000000000000000	0000000000000000	
M^*	309F4C5E31CAD0EE	0000000000000000	CF7CA0BD904331CB	
ΔM	309F4C5E31CAD0EE	0000000000000000	CF7CA0BD904331CB	
S_3	8B040660A8F0C7BF	09EE0D5A362F769E	B62FDC8118D186F2	96E6A8E0049B4BA7
	5494AA985B53A83F	B91DE273FA61A073	8082BCD3BB503820	56225FFB45DBA4F8
S_3^*	8B040660A8F0C7BF	09EE0D5A362F769E	B62FDC8118D186F2	96E6A8E0049B4BA7
	5494AA985B53A83F	B91DE273FA61A073	8082BCD3BB503820	56225FFB45DBA4F8
ΔS_3	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000000
$H(256)$	DE6D957A627CEBBF	88326DBED4135BB0	2039C5411191AD47	A15703E5EA2E66A2
$H^*(256)$	DE6D957A627CEBBF	88326DBED4135BB0	2039C5411191AD47	A15703E5EA2E66A2
$\Delta H(256)$	0000000000000000	0000000000000000	0000000000000000	0000000000000000

4 A Collision Attack on Twister-512

In this section, we show how the semi-free-start collision attack on Twister-512 can be extended to the hash function. We first show how to construct collisions in the compression function of Twister-512 with a complexity of 2^{223} compression function evaluations. This collision attack on the compression function is then extended to a collision attack on the hash function. The extension is possible by combining a multicollision attack and a birthday attack on the checksum. The attack has a complexity of about 2^{252} evaluations of the compression function of Twister-512.

4.1 Collision Attack on the Compression Function

For the collision attack on the compression function of Twister-512 we can use the characteristic of the previous section in the last Maxi-Round (see Figure 6).

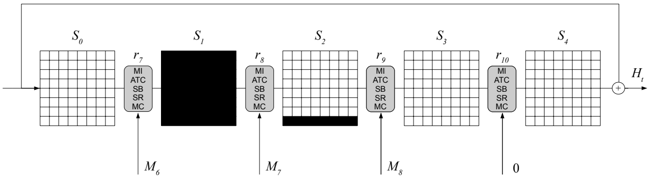


Fig. 6. The characteristic for the last Maxi-Round of Twister-512

Remember that in Twister-512 the 3 message words M_6, M_7 and M_8 are injected in the last Maxi-Round. Hence, we can use the first 5 message words $M_1 - M_5$ for a birthday match on 56 state bytes with a complexity of $2^{8 \cdot 56/2} = 2^{224}$. Since the 8 bytes of the last row can always be adapted by using the freedom in the (absolute) values of the message word M_6, we only need to match 56 out of 64 bytes. It can be summarized as follows:

1. Compute 2^{224} semi-free-start collisions for the last Maxi-Round of Twister-512 and save them in a list L. This has a complexity of about $3 \cdot 2^{224}$ Mini-Round computations. Note that we can choose from $2^{3 \cdot 64} = 2^{192}$ differences in M_6, M_7 and M_8 in the attack. Furthermore, by varying the values of M_7, we get additional 2^{64} degrees of freedom. Hence, we can construct up to 2^{256} semi-free-start collisions for the last Maxi-Round.
2. Compute the input of the last Maxi-Round by going forward and check for a match in the list L. After testing about 2^{224} candidates for the input of the last Maxi-Round we expect to find a match in the list L and hence, a collision for the compression function of Twister-512. Finishing this step of the attack has a complexity of about 2^{224} Mini-Round computations.

Hence, we can find a collision for the compression function of Twister-512 for the predefined initial value with a complexity of about 2^{223} compression function evaluations ($10 \cdot 2^{223}$ Mini-Round computations) and memory requirements of 2^{224}. Note that memory requirements of this attack can significantly be reduced by applying a memory-less variant of the meet-in-the-middle attack introduced by Quisquater and Delescaille [16], and applied by Morita, Ohata and Miyaguchi [12].

4.2 Collision Attack on the Hash Function

In this section, we show how the collision attack on the compression function can be extended to the hash function. The attack has a complexity of about 2^{252} evaluations of the compression function of Twister-512. Note that the hash function defines, in addition to the common iterative structure, a checksum computed over the outputs of each Mini-Round which is then part of the final hash computation. Therefore, to construct a collision in the hash function we have to construct a collision in the iterative structure (*i.e.* chaining variables) as well as in the checksum. To do this we use multicollisions.

A multicollision is a set of messages of equal length that all lead to the same hash value. As shown in [7], constructing a 2^t collision, *i.e.* 2^t messages consisting of t message blocks which all lead to the same hash value, can be done with a complexity of about $t \cdot 2^x$ for any iterated hash function, where 2^x is the cost of constructing a collision in the compression function. As shown in the previous section, collisions for the compression function of Twister-512 can be constructed with a complexity of 2^{223}. Hence, we can construct a 2^{256} collision with a complexity of about $256 \cdot 2^{223} \approx 2^{231}$ evaluations of the compression function of Twister-512 and memory requirements of 2^9 (needed to store the 2^{256} collision). With this method we get 2^{256} values for the checksum C that all lead to the same chaining value H_{256}.

To construct a collision in the checksum of Twister-512 we have to find 2 distinct messages consisting of 257 message blocks (256 message blocks for the multicollision and 1 message block for the padding) which produce the same value in the checksum. By applying a birthday attack we can find these 2 messages with a complexity of about 2^{256} checksum computations and memory requirements of 2^{256}. Due to the high memory requirements of the birthday attack, one could see this part as the bottleneck of the attack. However, the memory requirements can be significantly reduced by applying a memory-less variant of the birthday attack [16]. Hence, we can find a collision for Twister-512 with a complexity of about 2^{231} compression function evaluations (10 Mini-Rounds) and about 2^{256} checksum computations (8 xor operations and 8 modular additions of 64-bits). In general the cost for one checksum computation is smaller than one compression function evaluation. Depending on the implementation 1 checksum computation is $1/x$ compression function evaluation. Assume $x = 16$, then we can find a collision for Twister slightly faster than brute force search with a complexity of about 2^{252} compression function evaluations and negligible memory requirements.

4.3 A Remark on the Length Extension Property

Once, we have found a collision, *i.e.* collision in the iterative part (chaining variables) and the checksum, we can construct many more collisions by appending an arbitrary message block. Note that this is not necessarily the case for a straightforward birthday attack. By applying a birthday attack we construct a collision in the final hash value (after the output transformation Ω) and appending a message block is not possible. Hence, we need a collision in the iterative part as well as in the checksum for the extension property. Note that by combining the generic birthday attack and multicollisions, one can construct collisions in both parts with a complexity of about $256 \cdot 2^{256} = 2^{264}$ while our attack has a complexity of 2^{252}.

5 A Second-Preimage Attack on Twister-512

In this section, we present a second-preimage for Twister-512 with complexity of about 2^{384} compression function evaluations and memory requirements of 2^{64}. Assume we want to construct a second preimage for the message $m = m_1 \| \cdots \| m_{513}$. Then, the attack can be summarized as follows.

1. Construct a 2^{512} collision for the first 512 message blocks of Twister-512. This has a complexity of about $512 \cdot 2^{256} \approx 2^{265}$ compression function evaluations (using a birthday attack to construct a collision for each message block) and needs 2^{10} memory to save the multicollision. Hence, we get 2^{512} values for the checksum which all lead to the same chaining value H_{512}.
2. Choose an arbitrary value for the message block m_{513} with correct padding and compute H_{513}.
3. In the last iteration of the compression function, $H_{514} = f(H_{513}, C)$, we first choose arbitrary values for the five checksum words $C_1, \ldots, C_5$ with $C = C_1 \| \cdots \| C_8$ and compute the state $S_6^F = H_{513} \oplus S_3 \oplus S_6$. This also determines $S_{10} = H_{514} \oplus S_6^F$. Note that we know H_{514} from the first preimage.
4. For all 2^{64} choices of C_8 compute backward from S_{10} to the injection of C_7 and save the 2^{64} candidates for state $S_7' = \mathsf{MessageInjection}(\mathsf{S_7}, \mathsf{C_7})$ in the list L.
5. For all 2^{64} choices of C_6 compute forward from S_6 to the injection of C_7 and check for match of S_7' in the list L. Since we can still choose C_7, we only need to match 448 (out of 512) bits. In total, we get 2^{128} pairs and this step of the attack will succeed with probability $2^{-448+128} = 2^{-320}$. By repeating steps 3–5 about 2^{320} times we can find a match and fulfill this step of the attack with a complexity of about $2^{320+64} = 2^{384}$ compression function evaluations.
6. Once we have constructed a second-preimage for the iterative part, we still have to ensure that the value of the checksum C is correct. Therefore, we now use the fact that the checksum of Twister is invertible and we have 2^{512} values for the checksum which all lead to the same chaining value H_{512} and hence H_{513} and H_{514}. By using a meet-in-the-middle-attack, we can construct the needed value in the checksum. This has a complexity of about 2^{257} checksum computations and memory requirements of 2^{256}. Again the memory requirements can be significantly reduced by using a memory-less variant of the meet-in-the-middle attack [16].

Hence, we can construct a second-preimage for Twister-512 with complexity of about 2^{384} and memory requirements of $2^{10} + 2^{64}$. The memory requirements can be significantly reduced at the cost of an higher attack complexity. Several time/memory tradeoffs are possible between 2^{384+s} compression function evaluations and memory requirements of $2^{10} + 2^{64-s}$. Note that our attack requires that the first message consists of at least 513 message blocks. Due to the output transformation of Twister-512, the attack can not be extended to a preimage attack on Twister-512 in a straight-forward way.

6 A Preimage Attack on Twister-512

In order to construct a preimage, we have to invert the output transformation of Twister-512. Once we have inverted the output transformation, we can use the second preimage attack described in the previous section to construct a preimage

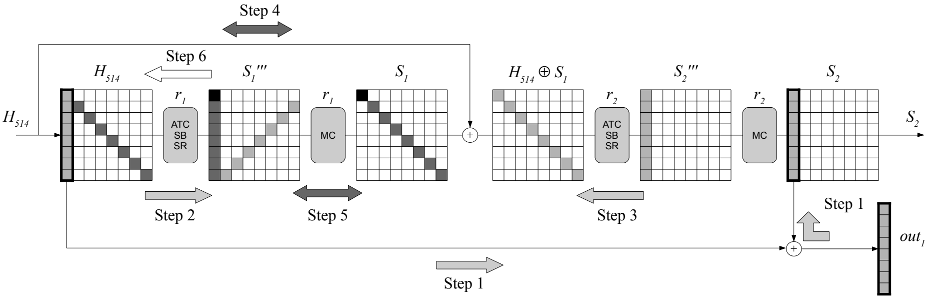


Fig. 7. The inversion of the first part of the output transformation of Twister

for Twister-512. Suppose we seek a preimage of $h = out_1 \| \cdots \| out_8$ consisting of 513 message block. Then we have to find the chaining value H_{514} such that $\Omega(H_{514}) = h$.

In the following, we show how to find a H_{514} such that out_1 is correct with a complexity of about 2^8 instead of the expected 2^{64}. This reduces the complexity of inverting the whole output transformation Ω to about 2^{456} instead of the expected complexity of 2^{512}. The inversion of the first step of the output transformation can be summarized as follows (see also Figure 7):

1. Choose a random value for the first column of H_{514}. Use out_1 and the first column of H_{514} (8 bytes each) to compute the first column of S_2 (8 bytes).
2. Compute the 8 bytes $S_1'''[i][1 + (9 - i) \mod 8]$ for $(1 \leq i \leq 8)$ of state S_1''' using the first column of H_{514}.
3. Compute backward through the Mini-Round r_2 for the first column of S_2 to get the diagonal 8 bytes of $S_1 \oplus H_{514}$.
4. Choose random values for the 8 diagonal bytes of H_{514}. Note that this determines the first column of S_1'''. Next, compute the 8 diagonal bytes of S_1 from the diagonal bytes of $H_{514} \oplus S_1$ and H_{514} using the feed-forward.
5. Now, we need to connect the states S_1''' and S_1 through the MixColumns operation of Mini-Round r_1. Note that the first column of S_1''' is already fixed (due to step 4). If the first byte of $S_1[1][1]$ does not match, we need to go back to step 1 again. After repeating steps 1-4 about 2^8 times we expect to find a match for $S_1[1][1]$. Once, we have found a match, we have to modify column 2-8 of S_1''' such that the remaining 7 bytes match as well.
   (a) For each column $i = 2 \ldots 8$ choose random values for the bytes $S_1'''[i][k]$ with $k \neq 10 - i$. Note that the bytes $S_1'''[i][k]$ with $k = 10 - i$ are already fixed due to step 2 of the attack.
   (b) Next, we compute the MixColumns operation and check if the byte $S_1[i][i]$ matches. If not, we repeat the previous step. This has a complexity of about 2^8.

   Since each column can be modified independently in the attack, finishing this step of the attack has a complexity of about $8 \cdot 2^8 \approx 2^{11}$ Mini-Round computations.

6. After we have found a match for all columns, we can compute backwards from S_1''' to determine H_{514}. Note that values fixed in step 1 and step 4 do not change anymore.

Hence, we can find a H_{514} such that out_1 is correct with a total complexity of about 2^{11} Mini-Round computations, respectively 2^8 compression function evaluations. By repeating the attack about 2^{448} times, we can invert the output transformation of Twister-512 with a complexity of about $2^{438} \cdot 2^8 = 2^{456}$ compression function evaluations and memory requirement of 2^{10}.

Once we have inverted the output transformation, *i.e.* we have found the chaining value H_{514} such that $\Omega(H_{514}) = h$, we can apply the second preimage attack described in the previous section to construct a preimage for Twister-512 consisting of 513 message blocks. The attack has a complexity of about $2^{448} + 2^{456} \approx 2^{456}$ compression function evaluations and negligible memory requirements.

7 Conclusion

In this paper, we have shown two main results: Although Twister is heavily based on a Merkle-Damgård style iteration (as many other hash function like SHA-2), the corresponding reduction proof that reduces the collision resistance of the hash function to the collision resistance of the compression function is not applicable anymore. We show practical (in time and memory) attacks and give an example for a compression function collision. This clearly invalidates the collision resistance assumption of the compression function.

Secondly, we give a theoretical collision, second preimage and preimage attack on the hash function Twister-512. Although the practicality of the proposed attacks might be debatable, it nevertheless exhibits non-random properties that are not present in SHA-512.

Acknowledgements

The authors wish to thank the designers of Twister for useful comments and discussions. The work in this paper has been supported in part by the European Commission under contract ICT-2007-216646 (ECRYPT II) and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

References

1. Barreto, P.S.L.M., Rijmen, V.: The WHIRLPOOL Hashing Function. Submitted to NESSIE (September 2000) (Revised May 2003), `http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html` (2008/07/08)
2. Brassard, G. (ed.): CRYPTO 1989. LNCS, vol. 435. Springer, Heidelberg (1990)
3. Damgård, I.: A Design Principle for Hash Functions. In: Brassard [2], pp. 416–427

4. Fleischmann, E., Forler, C., Gorski, M.: The Twister Hash Function Family. Submission to NIST (2008)
5. Fleischmann, E., Forler, C., Gorski, M., Lucks, S.: Twister - A Framework for Secure and Fast Hash Functions. In: Li, H., Bao, F. (eds.) ISPEC. Springer, Heidelberg (to appear, 2009)
6. Gauravaram, P., Kelsey, J.: Linear-XOR and Additive Checksums Don't Protect Damgård-Merkle Hashes from Generic Attacks. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 36–51. Springer, Heidelberg (2008)
7. Joux, A.: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)
8. Mendel, F., Pramstaller, N., Rechberger, C.: A (Second) Preimage Attack on the GOST Hash Function. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 224–234. Springer, Heidelberg (2008)
9. Mendel, F., Pramstaller, N., Rechberger, C., Kontak, M., Szmidt, J.: Cryptanalysis of the GOST Hash Function. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 162–178. Springer, Heidelberg (2008)
10. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) Fast Software Encryption. Springer, Heidelberg (to appear, 2009)
11. Ralph, C.M.: One Way Hash Functions and DES. In: Brassard [2], pp. 428–446
12. Morita, H., Ohta, K., Miyaguchi, S.: A Switching Closure Test to Analyze Cryptosystems. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 183–193. Springer, Heidelberg (1992)
13. National Institute of Standards and Technology. FIPS PUB 197, Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, U.S. Department of Commerce (November 2001)
14. Government Committee of Russia for Standards. GOST 34.11-94, Gosudarstvennyi Standard of Russian Federation, Information Technology Cryptographic Data Security Hashing Function (in Russian) (1994)
15. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl - a SHA-3 candidate (2008), `http://www.groestl.info`
16. Quisquater, J.-J., Delescaille, J.-P.: How Easy is Collision Search. New Results and Applications to DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 408–413. Springer, Heidelberg (1990)

A MixColumns and Inverse MixColumns

The MDS matrix A of the MixColumns operation of Twister and its inverse B are given as follows:

$$A = \begin{pmatrix} \mathsf{02} & \mathsf{01} & \mathsf{01} & \mathsf{05} & \mathsf{07} & \mathsf{08} & \mathsf{06} & \mathsf{01} \\ \mathsf{01} & \mathsf{02} & \mathsf{01} & \mathsf{01} & \mathsf{05} & \mathsf{07} & \mathsf{08} & \mathsf{06} \\ \mathsf{06} & \mathsf{01} & \mathsf{02} & \mathsf{01} & \mathsf{01} & \mathsf{05} & \mathsf{07} & \mathsf{08} \\ \mathsf{08} & \mathsf{06} & \mathsf{01} & \mathsf{02} & \mathsf{01} & \mathsf{01} & \mathsf{05} & \mathsf{07} \\ \mathsf{07} & \mathsf{08} & \mathsf{06} & \mathsf{01} & \mathsf{02} & \mathsf{01} & \mathsf{01} & \mathsf{05} \\ \mathsf{05} & \mathsf{07} & \mathsf{08} & \mathsf{06} & \mathsf{01} & \mathsf{02} & \mathsf{01} & \mathsf{01} \\ \mathsf{01} & \mathsf{05} & \mathsf{07} & \mathsf{08} & \mathsf{06} & \mathsf{01} & \mathsf{02} & \mathsf{01} \\ \mathsf{01} & \mathsf{01} & \mathsf{05} & \mathsf{07} & \mathsf{08} & \mathsf{06} & \mathsf{01} & \mathsf{02} \end{pmatrix}$$

$$B = A^{-1} = \begin{pmatrix} \mathsf{3E} & \mathsf{C5} & \mathsf{7A} & \mathsf{E7} & \mathsf{1B} & \mathsf{A9} & \mathsf{8A} & \mathsf{23} \\ \mathsf{23} & \mathsf{3E} & \mathsf{C5} & \mathsf{7A} & \mathsf{E7} & \mathsf{1B} & \mathsf{A9} & \mathsf{8A} \\ \mathsf{8A} & \mathsf{23} & \mathsf{3E} & \mathsf{C5} & \mathsf{7A} & \mathsf{E7} & \mathsf{1B} & \mathsf{A9} \\ \mathsf{A9} & \mathsf{8A} & \mathsf{23} & \mathsf{3E} & \mathsf{C5} & \mathsf{7A} & \mathsf{E7} & \mathsf{1B} \\ \mathsf{1B} & \mathsf{A9} & \mathsf{8A} & \mathsf{23} & \mathsf{3E} & \mathsf{C5} & \mathsf{7A} & \mathsf{E7} \\ \mathsf{E7} & \mathsf{1B} & \mathsf{A9} & \mathsf{8A} & \mathsf{23} & \mathsf{3E} & \mathsf{C5} & \mathsf{7A} \\ \mathsf{7A} & \mathsf{E7} & \mathsf{1B} & \mathsf{A9} & \mathsf{8A} & \mathsf{23} & \mathsf{3E} & \mathsf{C5} \\ \mathsf{C5} & \mathsf{7A} & \mathsf{E7} & \mathsf{1B} & \mathsf{A9} & \mathsf{8A} & \mathsf{23} & \mathsf{3E} \end{pmatrix} .$$

Cryptanalysis of CubeHash

Eric Brier and Thomas Peyrin

Ingenico, France
eric.brier@ingenico.com,
thomas.peyrin@ingenico.com

Abstract. CubeHash is a family of hash functions submitted by Bernstein as a SHA-3 candidate. In this paper, we provide two different cryptanalysis approaches concerning its collision resistance. Thanks to the first approach, related to truncated differentials, we computed a collision for the CubeHash-1/36 hash function, i.e. when for each iteration 36 bytes of message are incorporated and one call to the permutation is applied. Then, the second approach, already used by Dai, much more efficient and based on a linearization of the scheme, allowed us to compute a collision for the CubeHash-2/4 hash function. Finally, a theoretical collision attack against CubeHash-2/3, CubeHash-4/4 and CubeHash-4/3 is described. This is currently by far the best known cryptanalysis result on this SHA-3 candidate.

Keywords: hash functions, CubeHash, collision.

1 Introduction

A cryptographic hash function is a very important tool in cryptography, used in many applications such as digital signatures, authentication schemes or message integrity. One of its main and most important security feature is the collision resistance: finding two messages M and M' leading to the same hash value should require at least $2^{n/2}$ operations, where n is the output length of the hash function. Wang *et al.* [14,16,15,17] recently showed that most standardized hash functions (e.g. MD5 [12] or SHA-1 [10]) are not collision resistant. As a response, the National Institute of Standards and Technology (NIST) opened a public competition [9] to develop a new cryptographic hash algorithm that will be called SHA-3. 51 submissions met the minimum submission requirements, and had been accepted as the first round candidates. Among them, CubeHash is a new hash function designed by Bernstein [3] and currently under evaluation. One of its advantages is its simplicity of description which makes the analysis much easier for a cryptanalyst. This proposal can be considered as a stream-cipher oriented hash function. It maintains a big 1024-bit internal state, in which b-byte of message are incorporated at each iteration. After adding such a message block, a fixed permutation is then applied r time. Generally, the bigger is r (or the smaller is b) the harder it should be for the attacker to break the collision resistance of CubeHash-r/b.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 354–368, 2009.

Previous results. The first analysis of CubeHash was proposed by Aumasson *et al.* [1,2] in which the authors showed some non-random properties for several versions of CubeHash as well as a collision for CubeHash-2/120. Later, thanks to a linearization approach, Dai [6] computed a collision for CubeHash-1/45 and CubeHash-2/89. Those results were soon improved to CubeHash-2/12 [7].

Our contributions. In this paper, we provide two different cryptanalysis approaches concerning the collision resistance. The first approach, related to truncated differences, allowed us to compute a real collision for CubeHash-1/36. The second approach, i.e. linearizing the scheme, gives us much better results: we provide a real collision for CubeHash-2/4 and theoretical collision attacks against CubeHash-2/3, CubeHash-4/4 and CubeHash-4/3. This is currently the best known cryptanalysis result on CubeHash. To give an insight of the broken schemes, CubeHash-4/3 speed is about 26 cycles/byte, comparable to the speed of SHA-2 [10].

2 Description of CubeHash

We refer to the specifications of CubeHash [3] for the complete description of the scheme. CubeHash-r/b-h is the h-bit output version of CubeHash, for which b bytes of message are incorporated and r calls to the internal permutation F is applied at each iteration. The function can hash messages up to $2^{128} - 1$ bits. After an appropriate padding, the message is therefore divided into block M_i of b bytes each. A 1024-bit internal state is maintained, divided into 32 words X_i of 32 bits each and initialized by the following process: set the first three state words X_0, X_1, X_2 to the integers $h/8$, b, r respectively and set the remaining state words to 0. Then apply on the state $10 \times r$ times the internal permutation F.

After the initialization, the message blocks are treated iteratively: exclusive or the incoming b-byte message block onto the first b bytes of the internal state and apply the iteration function, composed of r times the application of the internal permutation F. When all the message blocks have been treated, exclusive or the integer 1 to the state word X_{31} and apply $10 \times r$ times the internal permutation F without incorporating message blocks anymore. Finally, output the h first bits of the internal state.

The internal permutation F uses very simple operations: rotation, exclusive or, modular addition and word swapping. It is composed of 10 steps (represented graphically in Figure 1):

1. Add X_i into $X_{i\oplus 16}$, for $0 \leq i \leq 15$.
2. Rotate X_i on the left by seven bits, for $0 \leq i \leq 15$.
3. Swap X_i and $X_{i\oplus 8}$, for $i \in [0, 1, 2, 3, 4, 5, 6, 7]$.
4. Xor $X_{i\oplus 16}$ into X_i, for $0 \leq i \leq 15$.
5. Swap X_i and $X_{i\oplus 2}$, for $i \in [16, 7, 20, 21, 24, 25, 28, 29]$.
6. Add X_i into $X_{i\oplus 16}$, for $0 \leq i \leq 15$.
7. Rotate X_i on the left by eleven bits, for $0 \leq i \leq 15$.

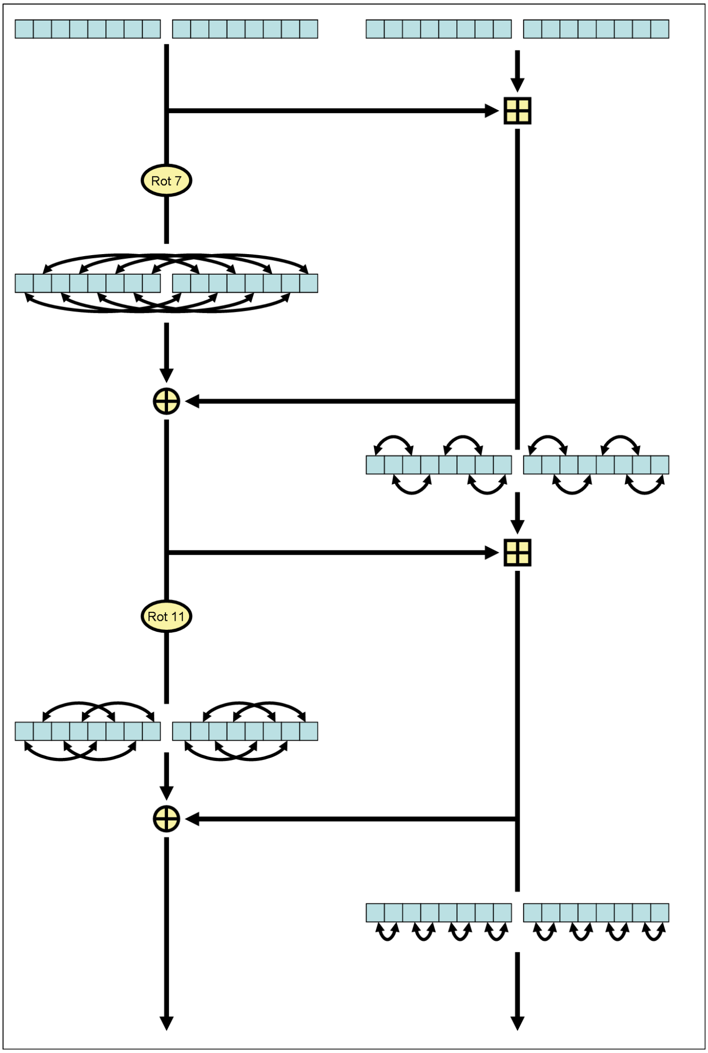


Fig. 1. Internal permutation F in `CubeHash`. Each cell represents a 32-bit word.

8. Swap X_i and $X_{i\oplus 4}$, for $i \in [0, 1, 2, 3, 8, 9, 10, 11]$.
9. Xor $X_{i\oplus 16}$ into X_i, for $0 \leq i \leq 15$.
10. Swap X_i and $X_{i\oplus 1}$, for $i \in [16, 18, 20, 22, 24, 26, 28, 30]$.

3 Truncated Differential Paths

In this section, we depict a cryptanalysis approach for `CubeHash` regarding its collision resistance, based on two very generic differential paths. The technique is related to truncated differences, originally introduced by Knudsen [8] to cryptanalyze block ciphers and later utilized by Peyrin [11] in the hash functions setting. These one-round differential paths can potentially be used to build more complex differential characteristics on any number of rounds per iteration. Our results provide theoretical collision attacks for CubeHash-$2/b$ with $r \leq 2$ and $b \geq 36$ and as a proof of concept, we computed a collision for the 512-bit version of CubeHash-$1/36$.

3.1 The Differential Paths

In the following, we say that a 32-bit word is *active* when a non-zero difference exists on this word. We denote a differential path for the internal permutation by $A \longmapsto B$, where A and B are 32-bit words for which each bit represents one internal word (the LSB will denote X_{31} and the MSB will denote X_0). Said in other words, we only check if the internal words are active or not, whatever are the values of the non-zero differences. For example, 0x05000000 ⟼ 0x00000800 means that we have X_5 and X_7 active on the input, and that we expect only X_{20} to be active on the output of the permutation.

In this paper, we use two distinct differential paths for the internal permutation (see also Figure 2):

$$\Delta_1 \ : \ \texttt{0xa0800000} \longmapsto \texttt{0x0a020000}$$

$$\Delta_2 \ : \ \texttt{0x0a020000} \longmapsto \texttt{0xa0800000}$$

One can see that the two-round differential path composed of Δ_1 and Δ_2 has the interesting property that the input and output active words patterns are the same. In the difference mask 0xa0800000, only the nine first 32-bit words of the internal state are active. Therefore, we can look for an attack on CubeHash-r/36 by using the differential paths Δ_1 and Δ_2 successively. In this section, we consider that the attacker insert 9 32-bit words of message at each iteration, i.e. he has full control on $X_0, \ldots, X_8$ at the beginning of each iteration.

We denote by $X^{\lll y}$ the rotation of y positions on the left applied to the word X and by $\overline{X}$ the complement value of X. Also, X' will represent the second member of the pair when X is an active word. For each differential path Δ_1 and Δ_2, a system of four equations on 32-bit words must be satisfied:

System 1 (for Δ_1) :

$$\begin{aligned}
(X_{24} + X_8) \oplus X_0^{\lll 7} &= (X_{24} + X'_8) \oplus X'^{\lll 7}_0 \\
X_8 + [(X_{26} + X_{10}) \oplus X_2^{\lll 7}] &= X'_8 + [(X_{26} + X_{10}) \oplus X'^{\lll 7}_2] \\
X_0 + [(X_{18} + X_2) \oplus X_{10}^{\lll 7}] &= X'_0 + [(X_{18} + X'_2) \oplus X_{10}^{\lll 7}] \\
X_2 + [(X_{16} + X_0) \oplus X_8^{\lll 7}] &= X'_2 + [(X_{16} + X'_0) \oplus X'^{\lll 7}_8]
\end{aligned}$$

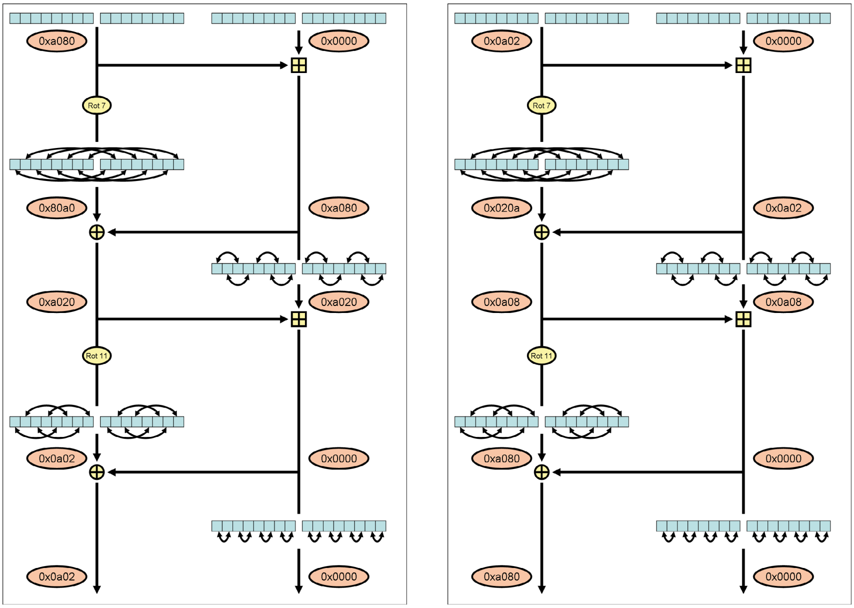


Fig. 2. Differential paths Δ_1 (left) and Δ_2 (right) for the internal permutation of `CubeHash`. A cell stands for a 32-bit word of the internal state and the boxes represent in hexadecimal display the active words during the computation.

System 2 (for Δ_2) :

$$\begin{aligned}
(X_{30}+X_{14}) \oplus X_6^{\lll 7} &= (X_{30}+X'_{14}) \oplus X'^{\lll 7}_6 \\
X_{14}+[(X_{28}+X_{12}) \oplus X_4^{\lll 7}] &= X'_{14}+[(X_{28}+X_{12}) \oplus X'^{\lll 7}_4] \\
X_6+[(X_{20}+X_4) \oplus X_{12}^{\lll 7}] &= X'_6+[(X_{20}+X'_4) \oplus X_{12}^{\lll 7}] \\
X_4+[(X_{22}+X_6) \oplus X_{14}^{\lll 7}] &= X'_4+[(X_{22}+X'_6) \oplus X'^{\lll 7}_{14}]
\end{aligned}$$

We describe here a fast method to resolve the first system of equations. Exactly the same method will also apply for the second system, so we will only focus on the first one. The internal state words X_{10}, X_{16}, X_{18}, X_{24} and X_{26} are given inputs of the system. The words X_0, X'_0, X_2, X'_2, X_8 and X'_8 are fully controlled by the attacker. Our method directly finds a solution for the three first equations. We will repeat this process several times until the last equation is also verified. More precisely, if we assume the equations to be independent, we will repeat the process 2^{32} times on average in order to get a solution for the complete system.

We now describe how to solve the three first equations of the system. First, we pick random values for X_2 and X'_2, such that $X_2 \neq X'_2$. We can rewrite the second and third equations respectively, so we directly get

$$X'_8 - X_8 = A \tag{1}$$

$$X'_0 - X_0 = B \tag{2}$$

where A and B are constant terms. We then set $Y = X_8 + X_{24}$ and $Y' = X'_8 + X_{24}$, so we can rewrite the first equation as:

$$Y \oplus X_0^{\lll 7} = Y' \oplus {X'}_0^{\lll 7}. \tag{3}$$

Finally, we combine the three equations together:

$$Y \oplus (A + Y) = X_0^{\lll 7} \oplus (B + X_0)^{\lll 7}. \tag{4}$$

We need a trick to solve this equation quickly: one can check that $x \oplus (x + k)$ is always equal to `0xffffffff` when $x = \overline{k}/2$ and when the least significant bit of k is equal to one. Thus, we wait for A and B values so that their least significant bit is equal to one and we set $Y = \overline{A}/2$ and $X_0 = \overline{B}^{\lll 25}/2$. The two sides of the equations are therefore equal to `0xffffffff` and we can finally deduce a solution that verifies the three first equations of our system.

3.2 A Collision for CubeHash-1/36

Using the differential paths Δ_1 and Δ_2 and the solving technique previously explained, we computed within a few minutes on an average PC (Processor Intel Core 2 Duo 2.0 GHz, with 2 GB of RAM) a collision for CubeHash-1/36 with 512 bits of output.

The collision attack uses four message inputs. The first message block is used without any difference in order to randomize the values of the internal state just after the initialisation (randomization of the equations systems). Then, the second message pair verifies the differential path Δ_1 and the third message pair the differential path Δ_2. Finally, the fourth and last message pair will erase the remaining differences (0xa0800000) in the internal state, and thus leads to an internal collision. Obviously, by adding some other message words without difference, one will maintain colliding pairs of internal states until the end of the computation of the hash function.

The values of the message words to insert and the final hash value are given in Table 4 in the Appendix A.

3.3 Extensions to Other Versions

One can easily extend this practical attack against CubeHash-1/36 to theoretical attacks against stronger versions. First, it is obvious that when $b \geq 36$, our attack remains valid. When $r = 2$, one has to verify equations from system 1 in the first internal permutation call, and equations from system 2 in the second internal permutation call of the iteration function.

We managed to generate input blocks verifying directly five equations among the eight of the two systems. Thus, one can assume that the three other equations will be verified with probability $2^{-32 \times 3}$. Therefore, one can find a collision for CubeHash-2/36 with a computation complexity of about 2^{96} function calls.

4 Linear Differential Paths

In this section, we will try to find interesting differential paths by using a $\mathcal{F}^2$-linear version of the scheme. More precisely, in the internal permutation F, we replace all the modular additions by bitwise XOR operations on 32-bit words. Overall, this simplification makes sense since non-linear components are quite few in `CubeHash`. This technique is very frequently used in order to study hash functions, for example in the case of `SHA-1` [4]. Dai [6,7] also used this simplification to compute a collision for `CubeHash`-2/12 and `CubeHash`-1/45.

4.1 The Differential Path

In this section, we are interested in attacking `CubeHash`-2/4. Said in other words, after a 2-round iteration, the attacker hash complete control over X_0 (and only X_0). The linear differential path we use is very simple, almost the simplest someone can think of. We first insert a one-bit perturbation in X_0 and apply the first 2-round iteration. We then erase all the differences present in X_0 and we apply the second 2-round iteration. Finally, we get a collision on the internal state by erasing a very last difference in X_0.

More precisely, we first insert a message pair with a one-bit difference located at bit position y. Then, the next message pair will correct three differences located at bit position $y + 4$, $y + 14$ and $y + 22$ (the positions are to be taken modulo 32). Finally, the last message pair will correct the last difference located at bit position $y + 4$. The differential path is described in Figure 3.

In the linear model, a collision can be computed directly. However, as any differential path, for the real function there is a certain probability of success that a random message pairs fulfilling the input differential constraints will also fulfill the output differential constraints. This probability can often be translated to a set of sufficient conditions to verify. In our case, the conditions directly depict the expected linear behavior of the scheme. The real `CubeHash` uses two modular additions in each call of the internal permutation F, which are non-linear components because of the carry potentially created. Thus, two interesting situations can occur:

1. a perturbation at a certain bit position is added to another bit containing no difference (move).
2. a perturbation at a certain bit position is added to another bit containing a difference (correction).

Each such situation will lead to exactly one sufficient condition because we want the modular addition carry to be unaffected by the perturbation. Said in other words, when we add together two word pairs A, B and A', B', the number of conditions will be the hamming weight of $(A \oplus A') \vee (B \oplus B')$, where $\vee$ stands for the bitwise or operation. When all the conditions from situations 1 and 2 are verified through the entire differential path, we are assured that the scheme behaved completely linearly regarding the bit perturbations and we can finally get our collision.

In total, the differential path presented possesses 46 sufficient bit conditions (24 for the first iteration and 22 for the second). This means that by testing

it.	round	step	active bits	nb. cond.
1	1	M	M_1^0	
1	1		X_0^0	
1	1	1	X_0^0, X_{16}^0	1
1	1	2	X_0^7, X_{16}^0	
1	1	3	X_8^7, X_{16}^0	
1	1	4	X_0^0, X_8^7, X_{16}^0	
1	1	5	X_0^0, X_8^7, X_{18}^0	
1	1	6	$X_0^0, X_8^7, X_{16}^0, X_{18}^0, X_{24}^7$	3
1	1	7	$X_0^{11}, X_8^{18}, X_{16}^0, X_{18}^0, X_{24}^7$	
1	1	8	$X_4^{11}, X_{12}^{18}, X_{16}^0, X_{18}^0, X_{24}^7$	
1	1	9	$X_0^0, X_2^0, X_4^{11}, X_8^7, X_{12}^{18}, X_{16}^0, X_{18}^0, X_{24}^7$	
1	1	10	$X_0^0, X_2^0, X_4^{11}, X_8^7, X_{12}^{18}, X_{17}^0, X_{19}^0, X_{25}^7$	
1	2	1	$X_0^0, X_2^0, X_4^{11}, X_8^7, X_{12}^{18}, X_{16}^0, X_{17}^0, X_{18}^0, X_{19}^0, X_{20}^{11}, X_{24}^7, X_{25}^7, X_{28}^{18}$	8
1	2	2	$X_0^7, X_2^7, X_4^{18}, X_8^{14}, X_{12}^{25}, X_{16}^0, X_{17}^0, X_{18}^0, X_{19}^0, X_{20}^{11}, X_{24}^7, X_{25}^7, X_{28}^{18}$	
1	2	3	$X_0^{14}, X_4^{25}, X_8^7, X_{10}^7, X_{12}^{18}, X_{16}^0, X_{17}^0, X_{18}^0, X_{19}^0, X_{20}^{11}, X_{24}^7, X_{25}^7, X_{28}^{18}$	
1	2	4	$X_0^0, X_0^{14}, X_1^0, X_2^0, X_3^0, X_4^{11}, X_4^{25}, X_9^7, X_{10}^7, X_{16}^0, X_{17}^0, X_{18}^0, X_{19}^0, X_{20}^{11}, X_{24}^7, X_{25}^7, X_{28}^{18}$	
1	2	5	$X_0^0, X_0^{14}, X_1^0, X_2^0, X_3^0, X_4^{11}, X_4^{25}, X_9^7, X_{10}^7, X_{16}^0, X_{17}^0, X_{18}^0, X_{19}^0, X_{22}^{11}, X_{26}^7, X_{27}^7, X_{30}^{18}$	
1	2	6	$X_0^0, X_0^{14}, X_1^0, X_2^0, X_3^0, X_4^{11}, X_4^{25}, X_9^7, X_{10}^7, X_{16}^{14}, X_{20}^{11}, X_{20}^{25}, X_{22}^{11}, X_{25}^7, X_{27}^7, X_{30}^{18}$	12
1	2	7	$X_0^{11}, X_0^{25}, X_1^{11}, X_2^{11}, X_3^{11}, X_4^{22}, X_4^4, X_9^{18}, X_{10}^{18}, X_{16}^{14}, X_{20}^{11}, X_{20}^{25}, X_{22}^{11}, X_{25}^7, X_{27}^7, X_{30}^{18}$	
1	2	8	$X_0^4, X_0^{22}, X_4^{11}, X_4^{25}, X_5^{11}, X_6^{11}, X_7^{11}, X_{13}^{18}, X_{14}^{18}, X_{16}^{14}, X_{20}^{11}, X_{20}^{25}, X_{22}^{11}, X_{25}^7, X_{27}^7, X_{30}^{18}$	
1	2	9	$X_0^4, X_0^{14}, X_0^{22}, X_5^{11}, X_7^{11}, X_9^7, X_{11}^7, X_{13}^{18}, X_{16}^{14}, X_{20}^{11}, X_{20}^{25}, X_{22}^{11}, X_{25}^7, X_{27}^7, X_{30}^{18}$	
1	2	10	$X_0^4, X_0^{14}, X_0^{22}, X_5^{11}, X_7^{11}, X_9^7, X_{11}^7, X_{13}^{18}, X_{17}^{14}, X_{21}^{11}, X_{21}^{25}, X_{23}^{11}, X_{24}^7, X_{26}^7, X_{31}^{18}$	
2	1	M	$M_2^4, M_2^{14}, M_2^{22}$	
2	1		$X_5^{11}, X_7^{11}, X_9^7, X_{11}^7, X_{13}^{18}, X_{17}^{14}, X_{21}^{11}, X_{21}^{25}, X_{23}^{11}, X_{24}^7, X_{26}^7, X_{31}^{18}$	
2	1	1	$X_5^{11}, X_7^{11}, X_9^7, X_{11}^7, X_{13}^{18}, X_{17}^{14}, X_{21}^{25}, X_{24}^7, X_{25}^7, X_{26}^7, X_{27}^7, X_{29}^{18}, X_{31}^{18}$	10
2	1	2	$X_5^{18}, X_7^{18}, X_9^{14}, X_{11}^{14}, X_{13}^{25}, X_{17}^{14}, X_{21}^{25}, X_{24}^7, X_{25}^7, X_{26}^7, X_{27}^7, X_{29}^{18}, X_{31}^{18}$	
2	1	3	$X_1^{14}, X_3^{14}, X_5^{25}, X_{13}^{18}, X_{15}^{18}, X_{17}^{14}, X_{21}^{25}, X_{24}^7, X_{25}^7, X_{26}^7, X_{27}^7, X_{29}^{18}, X_{31}^{18}$	
2	1	4	$X_3^{14}, X_8^7, X_9^7, X_{10}^7, X_{11}^7, X_{17}^{14}, X_{21}^{25}, X_{24}^7, X_{25}^7, X_{26}^7, X_{27}^7, X_{29}^{18}, X_{31}^{18}$	
2	1	5	$X_3^{14}, X_8^7, X_9^7, X_{10}^7, X_{11}^7, X_{19}^{14}, X_{23}^{25}, X_{24}^7, X_{25}^7, X_{26}^7, X_{27}^7, X_{29}^{18}, X_{31}^{18}$	
2	1	6	$X_3^{14}, X_8^7, X_9^7, X_{10}^7, X_{11}^7, X_{23}^{25}, X_{29}^{18}, X_{31}^{18}$	8
2	1	7	$X_3^{25}, X_8^{18}, X_9^{18}, X_{10}^{18}, X_{11}^{18}, X_{23}^{25}, X_{29}^{18}, X_{31}^{18}$	
2	1	8	$X_7^{25}, X_{12}^{18}, X_{13}^{18}, X_{14}^{18}, X_{15}^{18}, X_{23}^{25}, X_{29}^{18}, X_{31}^{18}$	
2	1	9	$X_{12}^{18}, X_{14}^{18}, X_{23}^{25}, X_{29}^{18}, X_{31}^{18}$	
2	1	10	$X_{12}^{18}, X_{14}^{18}, X_{22}^{25}, X_{28}^{18}, X_{30}^{18}$	
2	2	1	$X_{12}^{18}, X_{14}^{18}, X_{22}^{25}$	3
2	2	2	$X_{12}^{25}, X_{14}^{25}, X_{22}^{25}$	
2	2	3	$X_4^{25}, X_6^{25}, X_{22}^{25}$	
2	2	4	X_4^{25}, X_{22}^{25}	
2	2	5	X_4^{25}, X_{20}^{25}	
2	2	6	X_4^{25}	1
2	2	7	X_4^4	
2	2	8	X_0^4	
2	2	9	X_0^4	
2	2	10	X_0^4	
3	1	M	M_3^4	

Fig. 3. Linear differential path for CubeHash-2/4 and CubeHash-2/3. The three first columns give in order the iteration number, the round number and the step number in the internal permutation. A step denoted M represents the active bits of the message block inserted. The fourth column provides the active bits, where X_i^j denotes the j-th bit of the internal word X_i. Finally, the number of conditions is given in the last column.

about 2^{46} different messages pairs, one has a rather good chance to find a valid candidate. However, this can be improved by carefully choosing the bit position on which the initial difference is inserted. Indeed, some conditions can be placed on bit position 31 and will therefore be always verified with probability 1. We give in Table 1 the number of bit conditions of the differential path from Figure 3, according to the bit position of the initial difference inserted.

Table 1. Number of conditions for the differential path from Figure 3, according to the bit position of the first difference inserted

bit position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
nb. conditions	46	46	46	46	46	46	41	46	46	46	46	46	46	38	46	46	46	43	46	46	41	46	46	46	32	46	46	46	46	46	46	35

Note also that valid candidate search speed-ups are very likely to exist. Indeed, the complexity cost here only takes in account the probability of the differential path. However, in hash functions cryptanalysis, the use of the available degrees of freedom can drastically improve the overall complexity of the attack. For example, it is easy to force some bits of the first inserted message word in order to validate with probability 1 the very first condition of the differential path. Also, one can try to validate some conditions of the first iteration and some conditions of the second iteration independently, by playing with the message word inserted just before the second iteration.

4.2 Collision Attack for `CubeHash`-2/4 and `CubeHash`-2/3

According to Table 1, using the differential path from Figure 3 on bit position 24 provides a collision attack on `CubeHash`-2/4 with an overall complexity of 2^{32} operations. An example of such a collision for the 512-bit output version of the scheme is given here. It required a few minutes of computation on an average PC (Processor Intel Core 2 Duo 2.0 GHz, with 2 GB of RAM). No specific search speed-up were used which leaves room for further improvements.

This colliding pair is composed of five message inputs. The two first message blocks are used without any difference in order to randomize the values of the internal state just after the initialisation. Then, the third message pair inserts the initial one-bit difference at position 24, and the fourth message pair permits to prepare the internal state for the second part of the differential path. Finally, the fifth and last message pair will erase the remaining difference in the first word of the internal state, and thus leads to an internal collision.

The values of the message words to insert and the final hash value are given in Table 5 in the Appendix B.

In the implementation of `CubeHash`, one can check that if 3 bytes are inserted at each iteration, they will be xored to the 3 first least significant bytes of X_0 (the

it.	round	step	active bits	nb. cond.
1	1	M	M_1^0	
1	1		X_0^0	
1	1	1-10	$X_0^0, X_2^0, X_4^{11}, X_8^7, X_{12}^{18}, X_{17}^0, X_{19}^0, X_{25}^7$	4
1	2	1-10	$X_0^4, X_0^{14}, X_0^{22}, X_5^{11}, X_7^{11}, X_9^7, X_{11}^7, X_{13}^{18}, X_{17}^{14}, X_{21}^{11}, X_{21}^{25}, X_{23}^{11}, X_{24}^7, X_{26}^7, X_{31}^{18}$	20
1	3	1-10	$X_0^4, X_0^{14}, X_0^{22}, X_2^4, X_2^{14}, X_2^{22}, X_4^1, X_4^{15}, X_4^{25}, X_8^{11}$ $X_8^{21}, X_8^{29}, X_{12}^0, X_{12}^8, X_{12}^{18}, X_{12}^{22}, X_{14}^{18}, X_{17}^4, X_{17}^{14}, X_{17}^{22}$ $X_{19}^4, X_{19}^{14}, X_{19}^{22}, X_{22}^{25}, X_{25}^{11}, X_{25}^{21}, X_{25}^{29}, X_{28}^{18}, X_{30}^{18}$	30
1	4	1-10	$X_0^4, X_0^8, X_0^{12}, X_0^{28}, X_5^1, X_5^{15}, X_5^{25}, X_7^1, X_7^{15}, X_7^{25}, X_9^{11}, X_9^{21}, X_9^{29}$ $X_{11}^{11}, X_{11}^{21}, X_{11}^{29}, X_{13}^0, X_{13}^8, X_{13}^{22}, X_{17}^4, X_{17}^{18}, X_{17}^{28}, X_{21}^1, X_{21}^7, X_{21}^{25}, X_{21}^{29}$ $X_{23}^1, X_{23}^{15}, X_{23}^{25}, X_{24}^{11}, X_{24}^{21}, X_{24}^{29}, X_{26}^{11}, X_{26}^{21}, X_{26}^{29}, X_{31}^0, X_{31}^8, X_{31}^{22}$	60
2	1	M	$M_2^8, M_2^{12}, M_2^{28}$	
2	1		$X_0^4, X_5^1, X_5^{15}, X_5^{25}, X_7^1, X_7^{15}, X_7^{25}, X_9^{11}, X_9^{21}, X_9^{29}, X_{11}^{11}, X_{11}^{21}$ $X_{11}^{29}, X_{13}^0, X_{13}^8, X_{13}^{22}, X_{17}^4, X_{17}^{18}, X_{17}^{28}, X_{21}^1, X_{21}^7, X_{21}^{25}, X_{21}^{29}, X_{23}^1$ $X_{23}^{15}, X_{23}^{25}, X_{24}^{11}, X_{24}^{21}, X_{24}^{29}, X_{26}^{11}, X_{26}^{21}, X_{26}^{29}, X_{31}^0, X_{31}^8, X_{31}^{22}$	
2	1	1-10	$X_0^4, X_2^4, X_4^{15}, X_8^{11}, X_{12}^0, X_{12}^8, X_{14}^0, X_{14}^8, X_{14}^{22}, X_{17}^4, X_{19}^4$ $X_{22}^7, X_{22}^{15}, X_{22}^{29}, X_{25}^{11}, X_{28}^0, X_{28}^8, X_{28}^{22}, X_{30}^0, X_{30}^8, X_{30}^{22}$	56
2	2	1-10	$X_5^{15}, X_7^{15}, X_9^{11}, X_{11}^{11}, X_{13}^{22}, X_{17}^{18}, X_{21}^{15}, X_{21}^{29}, X_{23}^{15}, X_{24}^{11}, X_{26}^{11}, X_{31}^{22}$	29
2	3	1-10	$X_{12}^{22}, X_{14}^{22}, X_{22}^{29}, X_{28}^{22}, X_{30}^{22}$	18
2	4	1-10	X_0^8	4
3	1	M	M_3^8	

Fig. 4. Linear differential path for CubeHash-4/4 and CubeHash-4/3. The three first columns give in order the iteration number, the round number and the step number in the internal permutation. A step denoted M represents the active bits of the message block inserted. The fourth column provides the active bits, where X_i^j denotes the j-th bit of the internal word X_i. Finally, the number of conditions is given in the last column. The display of this differential path has been simplified because of the big number of active bits.

first byte of a word is considered to be the least significant one). Thus, using the same differential path but by inserting the initial perturbation at bit position 0 gives us a collision attack against CubeHash-2/3 with only 2^{46} operations. Indeed, if one inserts the perturbation at bit position 0 (in the first byte of X_0), the first message correction after one iteration will occur on bit positions 4, 14 and 22 (all located in the three first bytes of X_0). Finally, after another iteration, the attacker will have to erase the difference located at bit position 4 (in the first byte of X_0) in order to get an internal collision. One can see that all the control needed by the attacker is located in the three first bytes of X_0. Thus, a collision for CubeHash-2/3 can be found with only 2^{46} operations.

4.3 Collision Attack for CubeHash-4/4 and CubeHash-4/3

One can use the same linearization technique to cryptanalyze CubeHash-4/4 and CubeHash-4/3. The differential path will be a little bit more complicated than

Table 2. Number of conditions for the differential path from Figure 4, according to the bit position of the first difference inserted

bit position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
nb. conditions	221	221	198	218	221	221	212	221	221	195	207	221	221	207	221	221	210	207	221	221	189	221	221	213	202	221	221	197	221	221	216	202

Table 3. Best found probability of success for linear differential paths, according to the parameters of the hash function (r and b) and the maximum number of iterations

r	b	max nb. it.	probability
1	64	3	2^{3}
	32	5	2^{32}
	16	5	
	8	5	
	4	5	
	2	7	2^{221}
	1	15	2^{1225}
2	64	3	2^{32}
	32	3	
	16	3	
	8	3	
	4	3	
	2	4	2^{221}
	1	8	2^{1225}
4	64	3	2^{189}
	32	3	
	16	3	
	8	3	
	4	3	
	2	4	2^{964}
	1	9	2^{2614}
8	64	3	2^{650}
	32	3	2^{830}
	16	3	
	8	3	2^{1009}
	4	3	
	2	5	2^{2614}
	1	5	

the one from Figure 3 and its probability of success will also be much lower. We give in Figure 4 the new differential path for `CubeHash`-4/4. The display has been simplified because of the big number of conditions. However, since we use the linear model, the entire set of sufficient conditions can be easily retrieved from the input differences.

As for the previous differential path, the number of conditions depends on the bit position of the first perturbation inserted (because the conditions on bit position 31 are automatically verified). We give in Table 2 the number of bit conditions of the differential path from Figure 4, according to the bit position of the initial difference inserted.

One can directly conclude that a collision attack requiring 2^{189} operations can be mounted on `CubeHash`-4/4 by selecting the bit position 20 for the first perturbation inserted. Concerning `CubeHash`-4/3, only the bit positions 4 to 11 can be selected to insert the first perturbation, so that the control required by the attacker only involves the three least significant bytes of the internal word

X_0. Thus, by choosing the bit position 9, one gets a collision attack against CubeHash-4/3 with only 2^{195} operations.

4.4 Others Versions of CubeHash

We can apply similar techniques to cryptanalyze other versions of CubeHash. We give in Table 3 the best possible linear differential path probabilities, according to the parameters of the hash function considered and the maximum number of iterations. One can check that for some slower versions of CubeHash the probabilities of success of the linear differential paths are too low. However, it may be possible to use more iterations and aim to build a more complex attack composed of the concatenation of several individual linear differential paths. Also, depending on the amount of degrees of freedom available, one may be able to improve the overall complexity of the attack.

5 Conclusion

In this paper, we provided two different cryptanalysis approaches that led to the computation of a real collision for CubeHash-1/36 and CubeHash-2/4. The linear differential paths also give a theoretical collision attack against CubeHash-2/3 in 2^{46} operations, against CubeHash-4/4 in 2^{189} operations and against CubeHash-4/3 in 2^{195} operations. Those complexities have to be compared to 2^{128} and 2^{256} for an ideal hash function of 256 and 512-bit output respectively.

Acknowledgments

The authors would like to thank Jean-Philippe Aumasson, Shahram Khazaei, Willi Meier and María Naya-Plasencia for the very interesting discussions on CubeHash.

References

1. Aumasson, J.-P.: Collision for CubeHash2/120-512. NIST mailing list, local link (2008)
2. Aumasson, J.-P., Meier, W., Naya-Plasencia, M., Peyrin, T.: Inside the hypercube. Cryptology ePrint Archive, Report 2008/486 (2008)
3. Bernstein, D.J.: CubeHash specification (2.b.1). Submission to NIST (2008)
4. Chabaud, F., Joux, A.: Differential collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (1998)
5. Cramer, R. (ed.): EUROCRYPT 2005. LNCS, vol. 3494. Springer, Heidelberg (2005)
6. Dai, W.: Collisions for CubeHash1/45 and CubeHash2/89 (2008)
7. Dai, W.: Collision for CubeHash2/12 (2009)
8. Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)

9. National Institute of Standards and Technology. Cryptographic Hash Algorithm Competition
10. National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard (August 2002)
11. Peyrin, T.: Cryptanalysis of Grindahl. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 551–567. Springer, Heidelberg (2007)
12. Rivest, R.L.: RFC 1321: The MD5 Message-Digest Algorithm (April 1992)
13. Shoup, V. (ed.): CRYPTO 2005. LNCS, vol. 3621. Springer, Heidelberg (2005)
14. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer [5], pp. 1–18 (2005)
15. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup [13], pp. 17–36
16. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer [5], pp. 19–35
17. Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks on SHA-0. In: Shoup [13], pp. 1–16

Appendix A: Collision for CubeHash-1/36-512

Both message instances given in Table 4 lead to the following hash output, in hexadecimal display:

```
fb5e5578 c020296a 9d66df51 c49031ba
12c1eb92 eb404187 0b02e344 d2c9b335
5b1d6afa 8ac26a39 2fa35d96 c684bc3d
d6ecbd6e f71339e3 ba35bd72 841af694
```

Table 4. Message words to insert during four iterations in order to get a collision for the 512-bit output version of CubeHash-1/36. The values are given in hexadecimal notation.

Iteration 1

word i	M_i	M'_i	$M_i \oplus M'_i$
0	9d45c73a	9d45c73a	00000000
1	f6106042	f6106042	00000000
2	7f3be941	7f3be941	00000000
3	2b58d9ed	2b58d9ed	00000000
4	b7923ded	b7923ded	00000000
5	fb187e3f	fb187e3f	00000000
6	fd5d2414	fd5d2414	00000000
7	e66ffb2c	e66ffb2c	00000000
8	367126de	367126de	00000000

Iteration 2

word i	M_i	M'_i	$M_i \oplus M'_i$
0	43f69ab4	bc09654b	ffffffff
1	00000000	00000000	00000000
2	e3c37da8	1c6319ad	ffa06405
3	00000000	00000000	00000000
4	58fdd79b	58fdd79b	00000000
5	b08456a3	b08456a3	00000000
6	00000000	00000000	00000000
7	765e25fb	765e25fb	00000000
8	edaea852	bddcae41	50720613

Iteration 3

word i	M_i	M'_i	$M_i \oplus M'_i$
0	00000000	00000000	00000000
1	00000000	00000000	00000000
2	00000000	00000000	00000000
3	00000000	00000000	00000000
4	1be9de4a	1fa91472	0440ca38
5	00000000	00000000	00000000
6	8912b045	6b0ea65f	e21c161a
7	00000000	00000000	00000000
8	00000000	00000000	00000000

Iteration 4

word i	M_i	M'_i	$M_i \oplus M'_i$
0	00000000	7d23e83d	7d23e83d
1	00000000	00000000	00000000
2	00000000	07e4687b	07e4687b
3	00000000	00000000	00000000
4	00000000	00000000	00000000
5	00000000	00000000	00000000
6	00000000	00000000	00000000
7	00000000	00000000	00000000
8	00000000	8e6c1d83	8e6c1d83

Appendix B: Collision for CubeHash-2/4-512

Both message instances given in Table 5 lead to the following hash output, in hexadecimal display:

```
220f6a8a 640870f4 2757873d 8f16bc80
0f5595fa a519aa37 2091d3f0 c1e86527
fe9fa656 de7d1cb7 b9c367b2 a06d6616
27aa321d d3fd2ec6 378d61d1 9a270371
```

Table 5. Message words to insert during fives iterations in order to get a collision for the 512-bit output version of CubeHash-2/4. The values are given in hexadecimal notation.

	M	M'	$M \oplus M'$
Iteration 1	72d9dcf5	72d9dcf5	00000000
Iteration 2	b835e32f	b835e32f	00000000
Iteration 3	05a4593f	04a4593f	01000000
Iteration 4	b897ebd7	a897ab97	10004040
Iteration 5	00000000	10000000	10000000

Collision Attack on Boole

Florian Mendel, Tomislav Nad, and Martin Schläffer

Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
Tomislav.Nad@iaik.tugraz.at

Abstract. Boole is a hash function designed by Gregory Rose and was submitted to the NIST Hash competition. It is a stream cipher based hash function which produces digests up to 512 bits. Different variants exist, namely Boole16, Boole32 and Boole64 where the number refers to word size in bits. Boole64 is considered as the official submission. In this paper we demonstrate a collision attack with complexity 2^{65} for the 64-bit variant and 2^{33} for the 32-bit variant. The amount of memory required is negligible. Since the attack on Boole32 is practical, we present an example for a collision.

1 Introduction

A hash functions maps an input of arbitrary finite length to an output of a fixed length. The basic security requirements for a cryptographic hash function are:

– *collision resistance* – it is computationally infeasible to find two different inputs, which hash to the same output.
– *second preimage resistance* – for a given input, it is computationally infeasible to find a second input with the same hash value.
– *preimage resistance* – for a given output of a hash function, it is computationally infeasible to find an input that hashes to that output.

Recently, the NIST hash function competition [1] started. In this public competition to find an alternative hash function to replace the SHA-1 and SHA-2 hash functions, many new designs have been proposed. In November 2008, round one has started and in total 51 algorithms were have been accepted. One of the submitted hash functions is Boole designed by Gregory Rose [2]. It is a stream cipher based design like PANAMA [3]. Boole is an expansion of the stream cipher Shannon [4] but is also influenced by other cryptographic primitives. Boole is a cryptographic primitive that can be used as a hash function, message authentication code (MAC) and a synchronous stream cipher.

In this paper we will describe a method to construct a collision for the Boole hash function. A collision occurs if two different messages result in the same hash value. Boole maps messages of arbitrary length to a hash result of 224, 256, 384 or 512 bits. A generic collision attack for the strongest version producing a 512 bit hash values requires about 2^{256} hash function computations. We will show that with our method a collision can be found with a complexity of less than 2^{65} state update transformations and negligible amount of memory.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 369–381, 2009.

2 Description of Boole

Boole operates on W-bit words, $W \in \{16, 32, 64\}$. We refer to Boole16, Boole32 and Boole64 if we need to distinguish between the different word sizes. The Boole hash function supports output lengths up to $8 \cdot W$ bits. The internal memory consists of a 16-word register R and three word accumulators, namely x, r and l. The register is a nonlinear feedback shift register and at the end an output filter function is applied. Boole consist of three phases: input phase, mixing phase and output phase. In the following we explain these phases in more detail.

2.1 Input Phase

In the input phase, the accumulators and register words are updated with the message words m_t. Each message word is used once in the input phase.

$$\begin{aligned} temp &= f_1(l^{(t)}) \oplus m_t \\ l^{(t+1)} &= temp \lll 1 \\ x^{(t+1)} &= x^{(t)} \oplus m_t \\ r^{(t+1)} &= (r^{(t)} \oplus temp) \ggg 1 \\ R_3^{(t+1)} &= R_3^{(t)} \oplus l^{(t+1)} \\ R_{13}^{(t+1)} &= R_{13}^{(t)} \oplus r^{(t+1)} \end{aligned} \tag{1}$$

Afterwards the whole message has been processed and the register is cycled:

$$\begin{aligned} R_i^{(t+1)} &= R_{i+1}^{(t)}, \text{ for } i = 1, \cdots, 14 \\ R_{15}^{(t+1)} &= f_1(R_{12}^{(t)} \oplus R_{13}^{(t)}) \oplus (R_0^{(t)} \lll 1) \\ R_0^{(t+1)} &= R_1^{(t)} \oplus f_2(R_2^{(t+1)} \oplus R_{15}^{(t+1)}) \end{aligned} \tag{2}$$

In Figure 1 we have drafted the update step of the input phase.

2.2 Mixing Phase

After the input phase, the bit length of the input data, the output length and accumulators are mixed into the register. By *length* we denote the length of the input in bits, represented as a 64-bit integer and split into W-bit words. h is the length of the resulting hash value. The mixing phase is applied twice and is accomplished as follows:

$$\begin{aligned} R_0 &= R_0 \oplus length \\ R_4 &= R_4 \oplus l \oplus h \\ R_i &= R_i \oplus l, \forall i \in \{7, 10, 13\} \\ R_i &= R_i \oplus x, \forall i \in \{5, 8, 11, 14\} \\ R_i &= R_i \oplus r, \forall i \in \{6, 9, 12, 15\} \end{aligned}$$

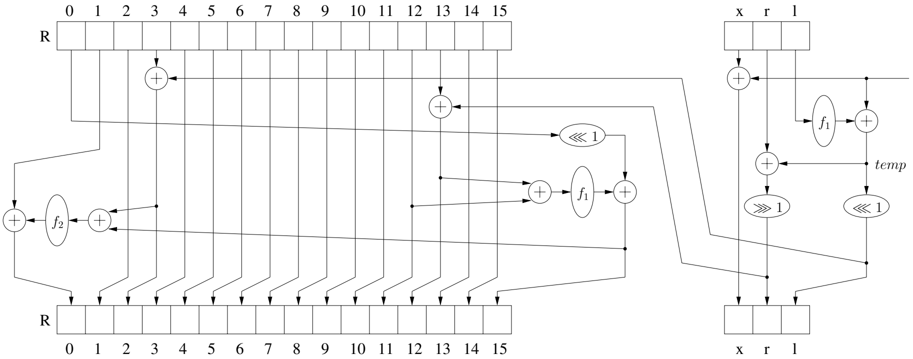


Fig. 1. Scheme of the update step

2.3 Output Phase

In the output phase, the content of the register and the output filter function is used to produce the hash value. First, the register is cycled as in Equation (2) and then, one word of the hash is computed as follows:

$$v = R_0 \oplus R_8 \oplus R_{12}$$

These steps are repeated until the required output length is reached.

2.4 Boolean Functions

The two nonlinear Boolean functions f_1 and f_2 depend on the the word size W. For Boole64 they are defined as follows:

$$\begin{aligned} t &= w \oplus \texttt{0x6996c53a} \\ t &= t \oplus ((t \lll C) \vee (t \lll D)) \\ t &= t \oplus ((t \lll B) \vee (t \lll E)) \\ t &= t \oplus ((t \ll A) \vee (t \lll F)) \end{aligned}$$

For $f_1(w) = t$ the constants $\{A, B, C, D, E, F\}$ are set to $\{3, 20, 34, 42, 55, 60\}$, and for $f_2(w) = t$ the constants $\{A, B, C, D, E, F\}$ are to $\{5, 27, 35, 46, 52, 55\}$. In the case of Boole32 the Boolean functions are defined as follows:

$$\begin{aligned} t &= t \oplus ((w \lll A) \vee (w \lll B)) \\ t &= t \oplus ((t \lll C) \vee (t \lll D)) \end{aligned}$$

For $f_1(w) = t$ the constants $\{A, B, C, D\}$ are $\{5, 7, 19, 22\}$ and for $f_2(w) = t$ the constants $\{A, B, C, D\}$ are $\{7, 22, 5, 19\}$. In Boole16, the Boolean functions are defined as follows:

$$\begin{aligned} t &= t \oplus ((w \lll A) \vee (w \lll B)) \\ t &= t \oplus ((\neg t \lll C) \vee (t \lll D)) \end{aligned}$$

For $f_1(w) = t$ the constants $\{A, B, C, D\}$ are $\{9, 13, 10, 15\}$ and for $f_2(w) = t$ the constants $\{A, B, C, D\}$ are $\{3, 14, 9, 10\}$.

3 A Differential Attack on Boole

In this section, we first analyze the differential properties of the components of Boole. We show that the Boolean functions f_1 and f_2 are not invertible and can be used to cancel differences. Then, we show how to find a collision in the accumulators and the register of Boole. Finally, we present a differential path which leads to a collision in the input phase. Since there are no message words used during the mixing and output phase, the collision in the input phase results in a collision of the full hash function Boole as well.

3.1 Collisions in the Boolean Functions

The Boolean functions f_1 and f_2 are used in every update step of the accumulator and the register of Boole. The main observation used in our attack is:

Observation 1. *The Boolean functions f_1 and f_2 are not invertible.*

Hence, we can find collisions in these functions and differences cancel out within the functions f_1 and f_2. In the following, we analyze which differences can be canceled and give the required conditions.

For Boole32 and Boole16 we get a zero output value for both f_1 and f_2 for the input values `0x0` and `0xF···F`. For Boole64 the input of the Boolean functions is first XORed with the constant `0x6996c53a`. Therefore, f_1 and f_2 collide for the values `0x6996c53a` and its inverted value `0x96693ac5`. The XOR difference for all variants of Boole is `0xF···F`. Note that there are more input values for f_1 and f_2 which collide. Table 1 shows all colliding input pairs with all-one difference for Boole32. Note that there are also more colliding input differences for the Boolean functions. However, in our attack we only use the all-one difference since this difference is rotation invariant and we can use the same difference in every step of Boole.

Table 1. Colliding input values for f_1 and f_2 with all-one difference for Boole32

w	$f_k(w)$	$f_k(w \oplus \texttt{0xFFFFFFFF})$
0x0	0x0	0x0
0x55555555	0x0	0x0
0xaaaaaaaa	0x0	0x0
0xFFFFFFFF	0x0	0x0

3.2 Difference Propagation in the Accumulator

In this section we show, how differences propagate and can be canceled in the accumulator. Whenever we injecting a message difference, we will first get a difference in all three accumulators x, r and l and the register words R_2 and

R_{12}. Remember that we can cancel the difference `0xF···F` in the function f_1 of the accumulator. Hence, the shortest differential path which leads to a collision in the accumulator is by injecting the same message difference `0xF···F` in two subsequent steps.

However, in this case the resulting differential path has a higher attack complexity. Therefore, we cancel the differences in the accumulator by injecting a second message difference after 3 steps. In this case, five differences are injected into the register.

The differences in the accumulators x and r are canceled by injecting the same difference in a subsequent message word. Whenever we inject the all-one difference using a message word, the resulting difference in the accumulator l is canceled using the function f_1 in the next step. According to Section 3.1, the difference `0xF···F` cancels if the input value of f_1 (l_t) is `0x0` for Boole32 and Boole16 or `0x6996c53a` for Boole64. If we inject the message difference `0xF···F` in step t, the following equation needs to hold for Boole64:

$$l^{(t+1)} = f_1(l^{(t)}) \oplus m_t = \texttt{0x6996c53a} \qquad (3)$$

Hence, the difference `0xF···F` in m_t will cancel in the following function f_1 if the value of m_t equals:

$$m_t = f_1(l^{(t)}) \oplus \texttt{0x6996c53a} \qquad (4)$$

3.3 The Differential Path

The full differential path, which leads to a collision in Boole is given in Table 2. Note that we only work with the all-one difference `0xF···F` in the whole path. This has a number of advantages. First, we can and do always cancel the difference in the functions f_1 and f_2. Second, whenever two differences are XORed, the resulting difference is zero. This is especially useful in the XOR prior to the functions f_1 and f_2, since we do not need any condition in these cases.

We inject the first message difference in message word m_3 since we need the previous message words to fulfill the conditions on the following functions f_1 and f_2 (see Section 5). We inject two differences into the register and cancel the differences in the accumulator using the message word m_6. Afterwards we have five differences in the register. By canceling input differences for the Boolean functions, the five differences are moving through the register and after 16 cycles they are again at the same positions. By injecting the same differences in the message words Δm_{19} and Δm_{22}, the five differences in the register are canceled. Hence, we get a collision in the register, accumulators and the full hash function Boole after 23 update steps.

Figure 3 of Appendix A shows the beginning (step 3-7) and Figure 4 shows the end of the differential path (`FF` denotes the all-one difference). From these figures it is easy to see in which step we need to cancel differences in f_1 and f_2 by defining conditions on the input. The last column of Table 2 lists all occurring non-zero input differences in f_1 and f_2 of the register.

Table 2. Difference propagation. I and II are modifications to cancel a difference in f_1. III cancels a difference in f_2.

t	$R_0^{(t)}$	$R_1^{(t)}$	$R_2^{(t)}$	$R_3^{(t)}$	$R_4^{(t)}$	$R_5^{(t)}$	$R_6^{(t)}$	$R_7^{(t)}$	$R_8^{(t)}$	$R_9^{(t)}$	$R_{10}^{(t)}$	$R_{11}^{(t)}$	$R_{12}^{(t)}$	$R_{13}^{(t)}$	$R_{14}^{(t)}$	$R_{15}^{(t)}$	$x^{(t)}$	$r^{(t)}$	$l^{(t)}$	m_t	m.m. type
0																					
1																					
2																					
3																				Δm_3	I,III
4			FF										FF				FF	FF	FF		
5		FF										FF	FF				FF	FF			
6	FF										FF	FF	FF				FF	FF		Δm_6	I
7			FF							FF	FF	FF				FF	-	-	FF		
8		FF							FF	FF	FF				FF						
9	FF							FF	FF	FF				FF							II,III
10							FF	FF	FF				FF			FF					II
11						FF	FF	FF				FF			FF						
12					FF	FF	FF				FF			FF							II
13				FF	FF	FF				FF			FF								II,III
14			FF	FF	FF				FF			FF									III
15		FF	FF	FF				FF			FF										III
16	FF	FF	FF				FF			FF											III
17	FF	FF				FF			FF							FF					III
18	FF				FF			FF							FF	FF					III
19				FF			FF							FF	FF	FF				Δm_{19}	
20			-			FF							-	FF	FF		FF	FF	FF		
21					FF								-	FF			FF	FF			
22				FF									-				FF	FF		Δm_{22}	
23			-														-	-	FF		
24																					

4 Message Modification

In the this section, we explain how to modify the message words to get a zero output difference in Boole. Message modification was introduced by Wang *et al.* in [5]. The basic idea of message modification is to use the degrees of freedom one has on the choice of the message words to fulfill conditions on the state variables.

In our attack we distinguish between three different types of message modification, depending on how the conditions for the inputs f_1 and f_2 occur. Note that it is more difficult to fulfill the conditions, if they occur for both Boolean functions in the same step or if a message difference is introduced in the same step.

4.1 Type I Message Modification

This type covers the situation where a non-zero input difference for f_1 occurs and a message difference is injected in the same step. In that case, we have to adept a previous message word to get a zero output difference. Figure 2 shows how the previous message word influences the input of f_1 and we get the following message modification equations:

$$\begin{aligned} x &= ((m_{t-1} \oplus f_1(l^{(t-1)}) \oplus r^{(t-1)}) \ggg 1 \oplus f_1(l^{(t)}) \oplus m_t) \ggg 1 \oplus R_{13}^{(t)} \\ y &= (m_{t-1} \oplus f_1(l^{(t-1)}) \oplus r^{(t-1)}) \ggg 1 \oplus R_{13}^{(t-1)} \end{aligned} \tag{5}$$

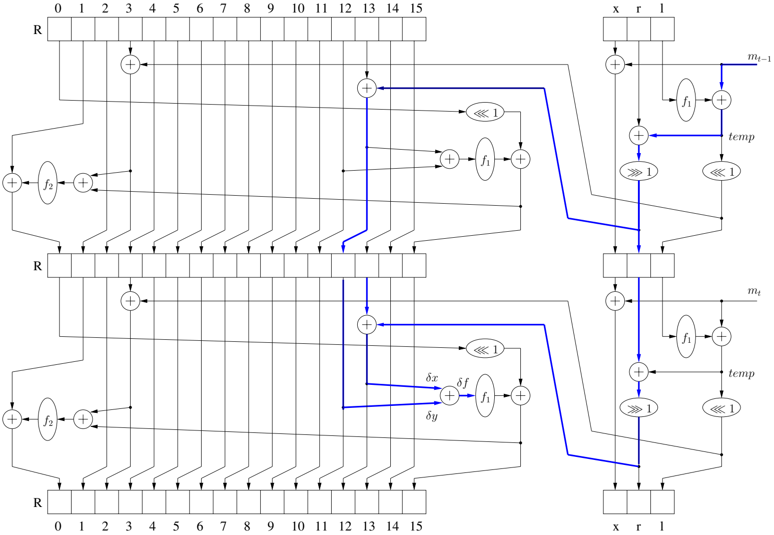


Fig. 2. Modification path for a collision in f_1

Hence, we have to find a message word m_{t-1} such that following equation holds:

$$x \oplus y = c,$$

where c is one of the values mentioned in Section 3.1. Instead of computing the message word itself, we compute the difference which is needed to change the current message word:

$$m_{t-1}^{\text{new}} = \delta m_{t-1} \oplus m_{t-1}$$

Then, equations (5) changes to

$$\delta x = \delta m_{t-1} \ggg 2$$
$$\delta y = \delta m_{t-1} \ggg 1.$$

Note that we ignore $f_1(l^{(t)}) \oplus m_t$, since it has always the same value, independent of the previous message words (see Section 3.2). We can then set up the following equation which expresses the needed difference for the input of f_1:

$$\delta f = \delta x \oplus \delta y = \delta m_{t-1} \ggg 2 \oplus \delta m_{t-1} \ggg 1 \tag{6}$$

For the value $c = 0$, δf is given by the following equation:

$$\delta f = R_{12}^{(t+1)} \oplus ((r^{(t)} \ggg 1) \oplus R_{13}^{(t)}) \tag{7}$$

Equation (6) defines a linear system of equations and $m_{t-1,j}$ denotes the jth bit of m_{t-1}:

$$\delta m_{t-1,i+1} = \delta m_{t-1,i} + \delta f_i \tag{8}$$

for $i = 0, \cdots W - 1$. To solve this system, we first choose a random value for $\delta m_{t-1,0}$. Then, we compute the remaining bits. Afterwards we check if the solution is correct by comparing

$$\delta m_{t-1,0} = \delta m_{t-1,W-1} + \delta f_{W-1}$$

to the randomly chosen value. A solution exists with probability 2^{-1}. If the solution is not correct we can choose a new message word m_{t-1}.

4.2 Type II Message Modification

The second case is much simpler and occurs if we have an input difference for f_1 in the register but we do not inject a message difference in the same step. Hence, we can achieve the needed input values for the Boolean function by modifying the message word in the same step t. The message is then computed as follows:

$$m_t = (R_{12}^{(t)} \oplus R_{13}^{(t)}) \lll 1 \oplus r^{(t)} \oplus f_1(l^{(t)})$$
$$m_t' = m_t$$

By this modification we get a zero output difference for f_1 with probability 1.

4.3 Type III Message Modification

For the case where a non-zero input difference for f_2 in step t occurs, we simply achieve a zero output difference by exhaustive search over all values of m_t or a previous message word, if in the same step also an other type of message modification has to be done.

5 The Collision Attack on Boole

In this section, all required steps to construct a collision for the Boole hash function, together with their complexities, are given.

1. The message words m_0, m_1 and m_2 are set to random values.
2. We inject a difference for m_3 and get a non-zero input difference for f_1 and f_2. We use type I message modification for f_1 and type III for f_2. Messages m_2 and m_0 are modified. The complexity of this step is 2^{W+1-d} update steps, where 2^d denotes the number of colliding input pairs for f_2 with all-one difference ($d = 2$ for Boole32).
3. Next we inject a difference in m_6 and get a condition for f_1. We solve this condition by type I modification of m_5. The complexity is about 2^1.
4. In step 9 we get again a non-zero input difference for f_2. A zero output difference is achieved by exhaustive search over all values of m_8. Additionally, a condition for f_1 is given which is solved by modifying m_9 according to type II message modification. The complexity of this step is 2^{W-d}.
5. In step 10 we get a non-zero input difference for f_1. We create a collision for f_1 by modifying m_{10} according to type II.
6. We do the same in step 12.
7. In step 13 we have conditions for f_1 and f_2. We do message modification of type II and III. For a zero output difference for f_2, m_{11} is used for the exhaustive search since m_{12} and m_{13} are already fixed. For each new value of m_{11}, m_{12} and m_{13} are recomputed. The complexity is again 2^{W-d}.
8. In step 14 we do an type II message modification of m_{14} to get a zero output difference for f_2. The same is done for step 15 and m_{15}, step 16 and m_{16}, step 17 and m_{17} and for step 18 and m_{18}. Each modification has a complexity of 2^{W-d}.
9. Finally, differences in m_{19} and m_{22} are injected which cancel all remaining differences.

The result is a collision in the register R and the accumulators x, r and l after the 23 step updates. Since all exhaustive searches are independent from each other, the overall attack complexity is given by $8 \cdot 2^{W-d} = 2^{W+3-d}$. For Boole64 this gives 2^{67-d} and we assume d to be at least 2. For Boole32 d is equal to two and therefore, the complexity is 2^{33} update steps.

5.1 Example Collision for Boole32

An example of two colliding message pairs for Boole32 is given in Table 3. The common hash value for both messages is

```
3f71dd7bd86ac4731bc1567791d6fc8479c411530e3c8230d97cbca36c19e01f.
```

Table 3. Two colliding messages for Boole32

m	a0bc0dbe	a1e5e09e	bcb01824	3403415f	0b177f21	7b31b82d	f5db2a23	a866bb7c
	004ebc0f	e11adc45	55b36c86	f59ed7ba	d7eb4405	c3265558	556eaf94	980d9839
	596fd2d9	d55ecff1	5df3155c	10dc14fa	22672d75	87fbd016	af0c15b8	4719bfdd
m'	a0bc0dbe	a1e5e09e	bcb01824	cbfcbea0	0b177f21	7b31b82d	0a24d5dc	a866bb7c
	004ebc0f	e11adc45	55b36c86	f59ed7ba	d7eb4405	c3265558	556eaf94	980d9839
	596fd2d9	d55ecff1	5df3155c	ef23eb05	22672d75	87fbd016	50f3ea47	4719bfdd

6 Conclusions

We presented a method to construct a collision for the Boole hash function. Boole was submitted to the NIST Hash competition, where the goal is to find a new secure hash algorithm (SHA-3). Boole is a stream cipher based design similar to PANAMA. However, we have shown in this paper, that Boole is not collision resistant. We are able to construct a collision in the internal register during the input phase. Since in the mixing and output phase no message inputs are used, this results in a collision for the whole hash function. In our attack we inject four message differences and have to modify a few messages words and after 23 steps the messages collide.

The main observation used in the attack is that the Boolean functions f_1 and f_2 are not invertible and we can construct collisions in these functions. The collision attack has a complexity of about 2^{W+3-d}, where W refers to the word size and 2^d the number of different colliding pairs for the Boolean functions f_1 and f_2. We provide an example of a colliding message pair for Boole32, since the attack complexity for this variant is about 2^{33} update steps and thus, feasible in practice.

Acknowledgements

The authors wish to thank Vincent Rijmen and the anonymous referees for useful comments and discussions. The work in this paper has been supported in part by the European Commission under contract ICT-2007-216646 (ECRYPT II) and by the Austrian Science Fund (FWF), project P19863.

References

1. National Institute of Standards and Technology: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register Notice (November 2007), http://csrc.nist.gov
2. Rose, G.G.: Design and primitive specification for boole. Submission to NIST (2008), http://seer-grog.net/BoolePaper.pdf
3. Daemen, J., Clapp, C.S.K.: Fast hashing and stream encryption with panama. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 60–74. Springer, Heidelberg (1998)
4. Hawkes, P., McDonald, C., Paddon, M., Rose, G., de Vries, M.W.: Design and primitive specification for shannon. IACR EPrint Archive (2007), http://eprint.iacr.org/2007/044
5. Wang, X., Yu, H.: How to break md5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)

A Differential Path for Boole

On the following pages, we show the beginning and the end of the differential path, which leads to a collision in the hash function Boole.

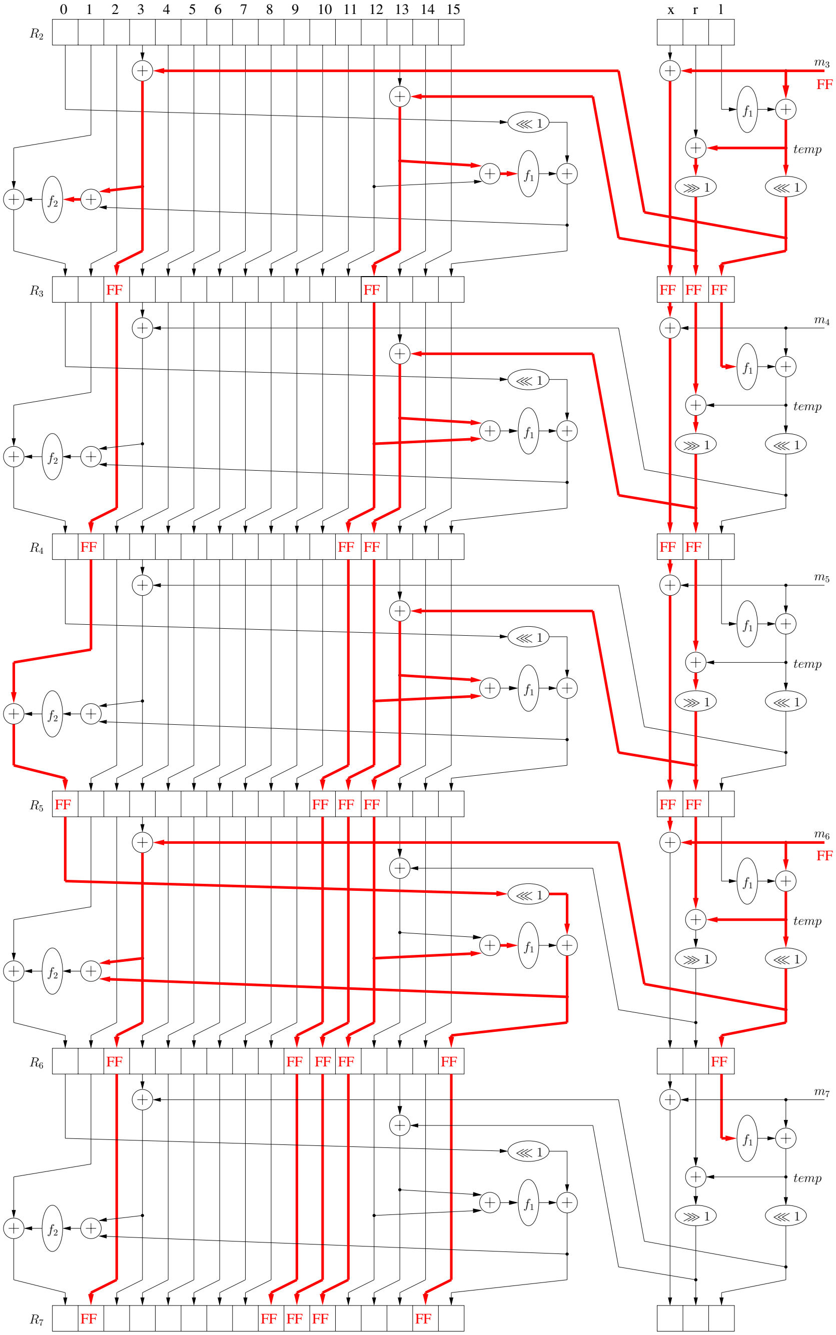


Fig. 3. Differential path for step 3 to 9

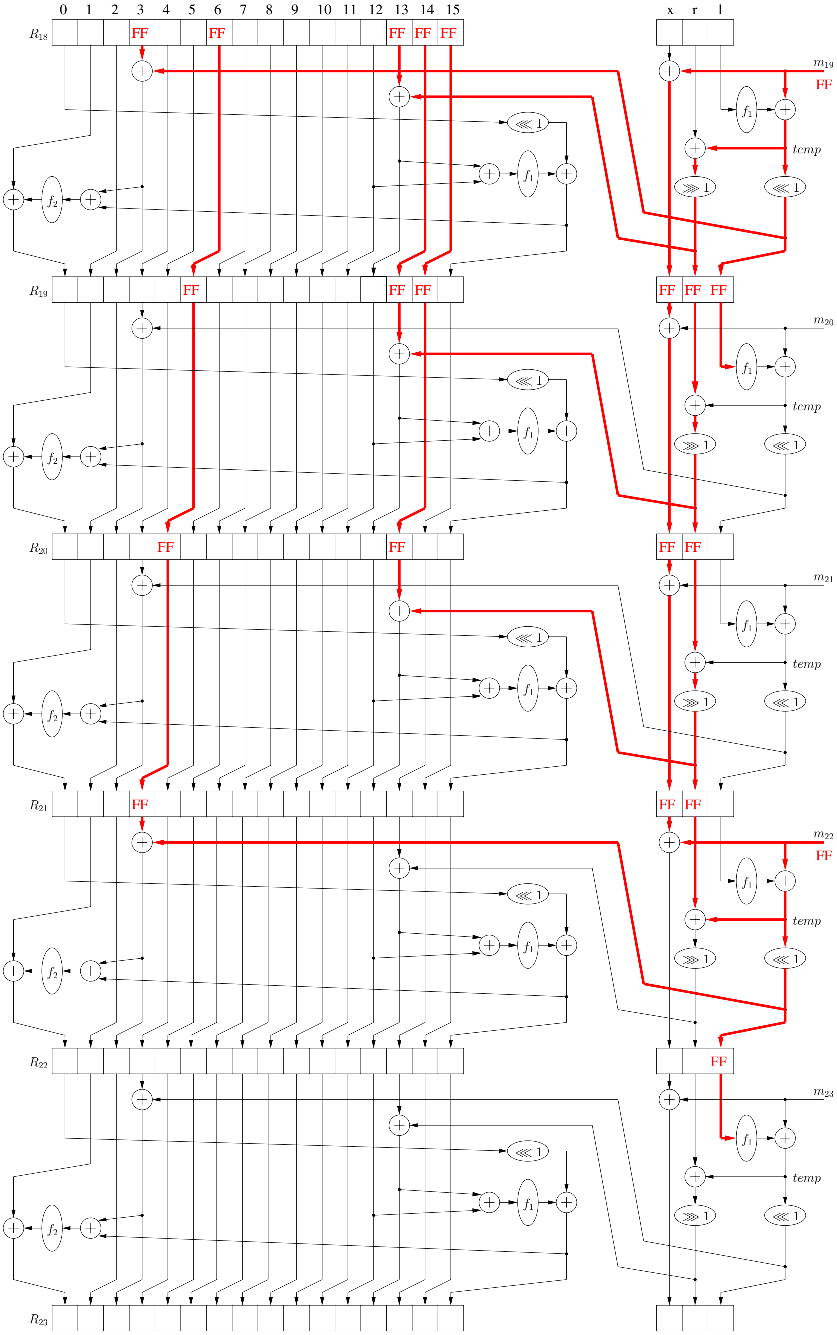


Fig. 4. Differential path for step 19 to 23

Integrity Protection for Revision Control

Christian Cachin[1] and Martin Geisler[2]

[1] IBM Research, Zurich Research Laboratory, Switzerland
cca@zurich.ibm.com
[2] Department of Computer Science, University of Aarhus, Denmark
mg@cs.au.dk

Abstract. Users of online-collaboration tools and network storage services place considerable trust in their providers. This paper presents a novel approach for protecting data integrity in revision control systems hosted by an untrusted provider. It guarantees atomic read and write operations on the shared data when the service is correct and preserves fork-linearizability when the service is faulty. A prototype has been implemented on top of the Subversion revision control system; benchmarks show that the approach is practical.

Keywords: Hash trees, memory checking, fork linearizability, storage security, applied cryptography.

1 Introduction

Nowadays people from all continents and all time zones collaborate together in global companies and other organizations, formal or not. Prominent examples are open-source development projects, such as the GNU/Linux operating system. For exchanging documents and storing the output of their work, they typically rely on a remote provider that hosts a shared storage service. An important class of such storage services are *revision control systems* (RCS) that facilitate collaboration on a set of documents that belong together and exist in multiple versions.

Although the collaborators trust the storage provider to preserve their documents, there are good reasons to verify that the provider indeed behaves correctly. For example, there are reported cases of break-ins to popular open-source repositories, where security-critical operating system code may have been altered undetectedly [7]. In cooperations that span multiple organizations, the storage provider often is a third party with little interest in the resulting work. Generally, verification reduces trust in the storage provider. To protect against faulty or corrupted storage providers, cryptographic protection methods are needed.

In this paper, we address *cryptographic integrity protection* for revision control systems. They represent the most important kind of multi-user storage and collaboration tools today, together with Wikis. We assume that clients are isolated and communicate directly with each other only under special circumstances; in fact, many clients may not even know each other. Our goal is to obtain a strong guarantee that a potentially faulty service provider has not altered the shared data.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 382–399, 2009.

The clients may use public-key signatures to authenticate their operations; this ensures that no unauthorized party can forge data in the repository. But in our model, replay attacks by a malicious storage server cannot be prevented, i.e., the server may return an outdated value to a reader, omitting a more recent update by another client. SUNDR [11] was the first storage system to address this problem by providing every client with a *fork-linearizable* view of the shared data. This notion ensures that all operations that a client does see are observed in the agreed linearization order, and if the server causes the views of two clients to differ in a single operation, they may never again see each others operations. This makes even subtle changes to the stored data easily detectable.

In this work, we describe the design and implementation of a consistent revision control system that preserves fork-linearizability. It relies on the fork-linearizable storage protocol of Cachin et al. [5] that reduces the communication overhead by an order of magnitude compared to the protocol of SUNDR. Our implementation extends the popular revision control system Subversion in a modular way.

The challenge in our work lies in the details of the integration of the fork-linearizable storage protocol with a revision control system. First, the abstract storage protocol uses only simple read and write operations on a file, whereas the revision control system implements transactions that usually read and update many files at once. Second, our goal is to be transparent to the server side of the underlying revision control system; therefore, we still rely on it to serialize concurrent updates. The implementation of our consistent revision control system merely extends this serialization order with the cryptographic consistency guarantees. Finally, the cryptographic operations must not be overly expensive; our hash-tree implementation exploits caching of the tree nodes and maintains them in the Subversion repository itself. This adds only little extra storage on top of the unchecked repository and requires few more operations.

1.1 Related Work

Protecting the integrity of stored data is an important question with a long history. But good solutions are needed today more than ever before [2], because personal and institutional data is stored and archived electronically. We describe here only a selection of the literature that uses the same model as our system, i.e., a remote untrusted bulk storage provider that offers read and write operations, accessed by one or more isolated clients with a small trusted memory.

Blum et al. [3] formalize the problem of memory checking and present the classical protection scheme based on hash trees [15]. With a memory consisting of n items and with random-access read and write operations to the memory, hash trees incur an overhead of $O(\log n)$ cryptographic operations. Several storage-system prototypes protect data integrity using hash trees; TDB [12] and SiRiUS [8] are two prominent examples. A similar approach has been proposed for protecting a CPU equipped with a trusted cache against unauthorized modifications to the main memory [6]. For database systems accessed through a query interface, Mykletun et al. [16] analyze the cost of integrity protection with cryptographic signatures that can be aggregated to reduce the space overhead. The recent work of Papamanthou et al. [17] shows how an array of data items can be authenticated with constant overhead for reading and sub-linear overhead for writing.

All systems mentioned so far consider either only one client or construct an abstraction of the trusted memory between clients (e.g., with digital signatures). The SUNDR system [11] is the only one protecting integrity for storage space shared by multiple clients that do not communicate among themselves. SUNDR guarantees linearizability when the storage service is correct and fork-linearizability when the service is faulty.

In distributed revision control, the two popular systems Git (`http://git.or.cz/`) and Mercurial (`http://www.selenic.com/mercurial/`) both employ hashes for identifying revisions. Without digital signatures, a corrupted server can trivially present modified changesets to a client (a changeset is the unit of an update between two revisions). The clients will no longer agree on the hashes identifying the revisions, but the server can keep passing content back and forth between the clients. Even if every client would sign all its updates, replay attacks would still be possible despite the use of hashes. Distributed revision control systems explicitly allow offline commits, and so the server can withhold changesets and claim that it has not seen them yet.

In practice, many open-source projects also publish digests or even cryptographic signatures on every release of their code. But since the cryptographic operations for authentication and verification are not transparently integrated with the storage mechanism, they require some manual intervention; hence, this method is not suitable for everyday collaboration.

1.2 Overview of the Paper

The remainder of the paper is organized as follows. Section 2 presents our system model and the design for our consistent revision control system. Section 3 describes our implementation. We evaluated our prototype system and present the results in Section 4. Section 5 discusses some limitations of our system and presents an outlook.

2 Design

This section presents the design of our consistent revision control system. In Section 2.1, we first describe the assumptions used by our system and the properties that it guarantees. We then introduce our abstract consistent storage service in Section 2.2, which provides a fork-linearizable storage space for small values, and review those properties of revision control systems that are relevant for our work in Section 2.3. In Section 2.4, we explain the design of the consistent revision control system.

2.1 Model

The system consists of an a priori unknown number of clients and a storage server. The server provides an abstraction of consistent shared storage to the clients, who access it using operations to read and write data, and with operations to control different revisions of the data. We assume that all clients are correct and follow the protocol. The server may be faulty or *corrupted* and deviate from the protocol in arbitrary ways, but not break any cryptographic primitives.

The clients never communicate with each other directly, they communicate only via the server. This model is convenient and realistic because the clients are not required to know each other, the network topology may prevent direct communication between them, and they can operate independently of each other. Revision control systems enable a convenient form of computer-supported cooperative work, because the collaborators can contribute at different times and from different locations.

We assume that each client is identified by a public key/private key pair, signed by a trusted certification authority (CA). Every client trusts one or more CAs, whose root keys it stores in a local directory in the form of self-signed X.509 certificates. Clients identify each other only by their public key; more precisely, clients accept every public key as the identity of another client when the key is accompanied by a certificate from a trusted CA. The system distributes the keys among clients as needed; a client only needs the trusted CA keys before it starts to interact with the storage service. Representing client identities by keys simplifies key distribution considerably [13].

Every client maintains a small trusted memory, whose size is independent of the size of the shared storage space. In order to prevent a corrupted server from introducing unauthorized modifications to the shared data, clients sign all their write operations and verify the integrity of the data they read using digital signatures. But since the clients do not communicate with each other, we cannot prevent that the server completes a write operation of one client, and still returns stale data to another client.

The notion of *fork-linearizability* provides the next-best notion of consistency in this model [14,5]. It ensures that all operations in the view of every client are legal in the sense that data returned by a read operation has been written by the indicated client, and that when the server causes the views of two clients to differ, even in a single operation only, then these clients may not see any further operation of each other afterwards.

Our goal is to implement a storage service that provides read and write operations, which execute atomically and according to their specification whenever the server is correct; when the server is faulty, the storage service still provides fork-linearizability. We refer to the work of Cachin et al. [5] for a formal notion that captures this requirement under the name of a *fork-linearizable emulation* of a storage service on a potentially corrupted server. In the subsequent sections, we explain how fork-linearizable storage is implemented by our consistent storage service and by our consistent revision control service.

Naturally, a corrupted server may simply refuse to cooperate, and then the clients will have to reconstruct the shared data from their own records. But this attack cannot be prevented. There is no easy solution to this problem, except to choose a more trustworthy server.

On the other hand, a fork-linearizable emulation ensures that the server cannot violate the consistency of the storage service and hide this attack from clients that are suspicious. Even if the clients communicate out-of-band only occasionally, for example, by sending email to each other directly, or through a discussion forum on a project website, they are guaranteed to immediately discover any inconsistencies that were introduced ever by a faulty server.

A more subtle attack occurs when the server conspires with a client and violates the assumption that clients are correct. The current design does not prevent such behavior, but our system provides some means that help the correct clients to recover from such attacks. We discuss these at the end of the paper (Section 5).

2.2 Consistent Storage Service

The *consistent storage service* (CSS) provides a simple interface for reading and writing short byte arrays and ensures fork-linearizability with an untrusted server. There is no hard limit on the size of the stored byte arrays, but the service is designed for sizes up to 10 or 100 KiB because all values are transiently kept in main memory.

CSS provides one storage location for every client, called a *register*. The client is the only one who may write to its register, but all clients may read from it, and there is an operation that reads all registers in a single step. Formally, CSS combines an array of single-writer/multi-reader registers [10] with an atomic snapshot object [1].

The service provides the following interface to clients, expressed as method invocations:

getkeys() returns a list of all client identities that are known to the server so far, represented by their public keys. The server learns the identity of a client as soon as the client invokes its first method. A client may use the output of the operation in subsequent queries.

write(*data*) stores *data* in the register of the client at the server, overwriting data previously stored there. In CSS, a client may only write to its own register.

read(*key*) reads the register identified by the public key *key* and returns the stored data. If no such data exists, the operation returns none.

readall() reads all registers in one step and returns a list of pairs (*key*, *data*), representing all registers stored by the server; every pair contains the corresponding client *key* and the stored *data*. This method is equivalent to invoking getkeys(), followed by invoking read(*key*) for all *key* values returned, all in one atomic step. Its purpose is to give a consistent view of all registers.

We use the *lock-step protocol* of Cachin et al. [5] to implement CSS. The protocol is noteworthy for using the server only as intermediary storage; in particular, the server does not perform any cryptographic operations. The protocol has been modified from using a fixed number of clients to handle an a priori unbounded number of clients that are identified only by a public key. Instead

Preliminaries. CSS stores a register value $data_{key}$ for each client identified by key. Only the client identified by key may write to $data_{key}$, but every client may read from any register. Every client locally maintains a timestamp that it increments during every operation. We call an array of timestamps a *version*; a version is an associative array V that maps keys to timestamps, denoted by $V[key] = t$. We write $V[key] = \bot$ if $V[key]$ is not defined. Versions acts as a vector clock for ordering operations. Two versions V and W are ordered so that V is *smaller than or equal to* W whenever $V[key] \leq W[key]$ for all values key such that $V[key] \neq \bot$.

Client state. The client maintains a version T representing its last completed operation. Note that a client identified by key finds its own timestamp in $T[key]$.

For simplicity of the description, we assume the client also keeps a copy of its own data value $data_{key}$ and writes it back during every read operation. (In the implementation, it only stores a collision-resistant hash of the data value and sends that in a read operation; in a write operation, it sends the data value.)

Server state. The server stores the register values in an associative array X, where entry $X[key]$ contains $(data_{key}, \sigma_{key})$, representing the register value and a digital signature issued under key on the string $\mathsf{value} \,\|\, data_{key} \,\|\, t$, where t is a timestamp equal to $T[key]$ when the client completed the operation that wrote $data_{key}$.

The server also keeps information from the last completed operation: the version V associated to it, the key $last$ identifying the client performing the operation, and a digital signature ω under key $last$ on $\mathsf{commit} \,\|\, V$.

Operation. When a client identified by key invokes a write, read, or readall operation, it sends the request together with key in a *submit message* to the server. The server sends a *reply message*, containing the version V, the key $last$, and the accompanying signature ω from the last operation. In a read operation for register identified by $rkey$, the server also sends the register value $X[rkey] = (data_{rkey}, \sigma_{rkey})$. In a readall operation, the server adds all register values X. The server then waits for a *commit* message from this client and does not process any messages from other clients.

The client verifies that the reply message contains valid data: the version V must be at least as large as its own version T, the entry $V[key]$ must be equal to its own timestamp $T[key]$, and the signature ω on $\mathsf{commit} \,\|\, V$ must be valid under key $last$. In a read or readall operation, the client also verifies that σ_{rkey} is a valid signature under $rkey$ on the string $\mathsf{value} \,\|\, data_{rkey} \,\|\, V[rkey]$, either for only one $rkey$ in a single-register read or for all values $rkey$ such that $X[rkey] \neq \bot$ in a readall operation. When the client detects any inconsistency in the reply, it considers the server to be faulty, generates an alarm, and aborts.

After the client has successfully verified the reply, it adopts the received version V as its own version T, increments its timestamp $T[key]$, and signs the new version T, resulting in a signature φ. It issues another signature σ on $\mathsf{value} \,\|\, data_{key} \,\|\, T[key]$, binding its data value to the timestamp. Then it sends a *commit message* to the server, containing T, φ, $data_{key}$, and σ.

When receiving the commit message, the server stores T, key, and φ as its version V, client key $last$, and signature ω that represent the last operation. The server also updates $X[key]$ with the received value $data_{key}$ and σ.

Fig. 1. The implementation of CSS using the lock-step protocol (adapted from [5,14])

of using vectors, versions are represented by an associative array that maps every known client key to the corresponding timestamp. The clients maintain some state in their local memory and save it on persistent storage between operations. The protocol is shown in Figure 1.

The lock-step protocol has the drawback of not being wait-free [10] because when the server waits for the *commit* message from a client, no other client can proceed with an operation. Mazières and Shasha [14] and Cachin et al. [5] both present seemingly more efficient protocols that allow some client operations to proceed in parallel. However, it has been shown that in all fork-linearizable storage emulation protocols, a reader must wait for a concurrent writer [5][1].

We therefore chose to implement CSS with the lock-step protocol for the following reasons: First, the addition of the readall operation introduces the above conflict between readall and *every* write operation. We know that our consistent revision control application (described in Section 2.4) will use only write and readall operations, and we expect that they occur about equally often. Hence, the potential for exploiting concurrency is reduced to concurrent read operations. Second, the protocol allowing for concurrent operations is considerably more involved than the lock-step protocol. The small potential gain did not merit the added implementation complexity.

2.3 Revision Control

A *revision control system* (RCS) provides operations for storing and retrieving multiple versions of the same set of documents. It facilitates collaboration among multiple users, who may work independently with the information. The RCS assigns revision numbers to the documents and maintains a history of all versions. The documents usually consist of a hierarchical set of files and directories in a file system. Revision control systems are an important collaboration tool, as can be seen from the large number of existing systems (Wikipedia's "List of revision control software" lists 64 systems as of Sept. 2008).

For the purpose of designing our consistent RCS, we describe here the main features of a *generic* centralized RCS. A *centralized* RCS uses a dedicated server for controlling revisions and storing the history, in contrast to a *distributed* RCS, where this task is shared by all users. Our RCS is modeled after two popular RCS for source code, CVS (`http://www.nongnu.org/cvs/`) and Subversion (`http://subversion.tigris.org/`); they both allow users to update the same document concurrently.

We expect the client interface of an RCS to provide the following main operations:

Checkout: A checkout operation transfers all documents from the server repository to the client. It creates a copy of the files and directories on the client, called the *working copy*. All editing takes place there. The RCS also supports attributes attached to documents and version control for them.

[1] Weaker semantics than fork-linearizability can give rise to wait-free storage emulation protocols [4].

Commit: After adding, modifying, or deleting some files in the working copy, the client wants to transfer the changes back to the central server, thereby making the changes visible to other clients. The client does this with a commit (or checkin) operation. Its effect is to create a new *revision* with a distinct identifier, called the *revision number*. We assume that revision numbers in a sequence of commits issued by multiple clients increase monotonically over time.

Update: An update operation transfers the most recent revision of all files from the server to the client and updates the working copy accordingly. The system also supports updating to a revision with a particular revision number.

When a client has modified some files locally and wants to commit the changes, it may have to perform an update, before the RCS allows a commit operation. This happens when some modifications of the client overlap and *conflict* with modifications committed by others. In this case, the commit operation will fail, the client is told to first update its working copy and to merge the concurrent changes, before the client may attempt another commit.

Typical RCS also support operations to populate the server repository with a set of documents initially, to rename repository contents, to create branches and merge them again, and to tag revisions with keywords. These operations may be present, but are not our main focus because they can be expressed as variations of the above three main operations.

We assume that all operations are transactional so that their changes either take effect in one atomic step on the server, or leave no trace in the repository in case of a failure.

2.4 Consistent Revision Control

Our *consistent revision control system* (CRCS) implements a revision control system that protects the integrity of the repository against a corrupted server. CRCS provides the same operations as an ordinary RCS and emulates a fork-linearizable storage service on the repository. We achieve fork-linearizability in terms of the checkout and update operations of CRCS, which implement a read operation on the repository, and in terms of the commit operation, which implements a write operation on the repository.

Fork-linearizability for a revision control system guarantees the following. Suppose a client A updates its working copy with CRCS to some revision number r. If A sees even a single file that was committed by another client B in revision r, then fork-linearizability implies that all files in client A's working copy have been cryptographically verified and are equal to those committed by B in revision r. Conversely, if there exists a more recent revision $s > r$ committed by a third client C, and the server hides revision s from A, then A can never again update to any revision committed by C or by anyone who updated to s. Because of this all-or-nothing implication of fork-linearizability, one can very easily detect even subtle modifications of a single file by a corrupted server.

We implement CRCS by combining our CSS with an unmodified RCS. CRCS computes a hash tree [15] over the set of documents in the repository and basically stores the root hash of the tree using CSS. This construction extends the integrity guaranteed by CSS from the root hash to the entire data. Suppose every client commits changes to CRCS by first committing its working copy using RCS, thereby obtaining a revision number r, computing the new root hash h, and then writing the tuple (r, h) to its register. This stores all information in CSS that another client needs for updating its working copy to the most recent revision and for verifying its integrity. But because CRCS also supports cryptographically verified update operations for previous revisions in the repository, the design is more complex.

Every client maintains a *revision log* L with information about every revision that it committed. The revision log is a list of tuples (r, h, c), denoting the revision number r, the root hash h, and a *revision commitment* c, sorted chronologically (i.e., according to r). Let H denote a collision-free cryptographic hash function. The revision commitment binds together all previous commit operations of the client in a hash chain; when committing revision r with hash h, the client computes c as $H(r \,\|\, h \,\|\, c')$, using the revision commitment c' from the last tuple in L (or $c' = \perp$ if L is empty). The same chaining scheme has been used in many other timestamping and data authentication algorithms [9].

For the description of the CRCS algorithm below, assume that every client stores its complete revision log in L. For increased efficiency, an implementation may actually maintain only the last tuple of L in CSS and keep the rest of L in untrusted shared storage; the collision resistance of H guarantees the uniqueness of every revision log given its last revision commitment.

The client proceeds now as follows to implement the main operations of CRCS. If one of the checks in the algorithm fails, the client generates an alarm and aborts.

Checkout: To check out the highest revision, the client invokes the readall() operation of CSS and determines the largest revision number r from the returned revision logs and the corresponding root hash h. After invoking checkout of RCS for revision r, the checkout algorithm recomputes the hash tree on the working copy and verifies that its root hash is equal to h.

Commit: The client first calls the update operation of CRCS (see below) to bring its working copy to the most recent revision according to CSS. Then it commits the working copy with RCS to obtain a new revision number r. If this fails, the operation aborts and the client is told to update and to try again. If all goes well, the client computes the root hash h of the hash tree on its working copy, extends the client's revision log L with r and h, and invokes write(L) from CSS.

Update: The update operation is very similar to checkout. The client performs readall() to obtain all revision logs, determines the largest revision number r with corresponding root hash h, calls update from RCS to bring its working copy to revision r, recomputes the changed paths in the hash tree, and verifies that the root hash matches h.

For updating to a particular revision r, the algorithm determines the client that committed r from all revision logs, locates the corresponding tuple $(r, h, \cdot)$ in some revision log L, and verifies L by following the hash chain from the tuple with r to the end of L. Then it proceeds as above, updating to revision r from RCS and verifying the working copy with respect to h.

When recomputing the hash tree for files that have changed in the repository, it is important that the client does that on a clean working copy, before the modifications from its working copy are applied. As the RCS merges the updates with the client's own changes, the update operation creates a working copy that differs from revision r in the repository.

Because the operations of CRCS verify that the working copy is consistent with the revision numbers and their root hashes maintained by CSS, the fork-linearizability of CSS implies the same property also for CRCS.

Note that the above algorithm introduces no new race conditions compared to RCS. As a consequence of synchronizing the client with CSS and RCS, it would be possible to create such problems. But the atomicity of the operations on CSS ensures that the more complex operations of CRCS are also atomic. In particular, whenever a client invokes checkout or update and retrieves some revision number from CSS, it always finds this revision in the repository of RCS. This holds because the commit operation of RCS precedes the writing of the corresponding revision number to CSS. Of course, there may already exist a more recent revision in the repository of RCS in the mean time, but this may also happen in the generic RCS, when another commit operation occurs immediately after an update.

3 Implementation

We have implemented our design in Python on Unix in two parts: first, the consistent storage service and, second, the consistent revision control system. The Python programming language encourages the kind of rapid prototyping we wanted and allowed a very natural transcription of the protocols. We chose *Subversion* (SVN) as the lower-level revision control system because it is widely used and because it fits our model of an RCS from Section 2.3. Hence, we refer to our implementation as *Consistent Subversion* (CSVN). Cryptographic operations are provided by OpenSSL via the M2Crypto Python interface to OpenSSL [18].

3.1 Consistent Storage Service

The implementation of CSS according to Section 2.2 stores arbitrary byte arrays. It is available as a library to clients. We wrote a simple interactive client application to read and write values entered by the user. The rich syntax of Python resulted in the server part of the algorithm in Figure 1 consisting of about 250 lines of code and the client part consisting of about 200 lines of code, including the operations for key management. Having a succinct implementation is important for maintainability, and especially important for security-relevant software.

CSS uses Python's object serialization over TCP connections for transport. The server implementation is single-threaded according to the lock-step protocol; it uses a time-out in order to tolerate a client that crashes between sending a submit message and sending the corresponding commit message. We plan to integrate SSL/TLS support for increasing the security of the client-server connections in the future; currently, network attacks appear to the clients as server faults.

3.2 Consistent Revision Control with Subversion

We implemented CSVN in the form of a library that interfaces to SVN and provides the three main revision control operations. The operations invoke our consistent storage service and the Python SVN Extension (`http://pysvn.tigris.org/`). The SVN server remains unchanged. We also created small wrapper scripts for a user to invoke the client operations. The CSVN library consists of about 170 lines of code, and the scripts of about 75 lines of code each. Hence, the code is very compact.

For the description below, let a *path* denote the unit of information managed by SVN; a path may be a directory containing other paths, a file, or a symbolic link.

Hash Trees. The protocol requires to compute a hash tree over the documents in a revision. Let us define a hash function $\mathcal{H}$ on paths maintained by SVN. The hash value of a path p that represents a file or a symbolic link is defined as

$$\mathcal{H}(p) = H\big(H(p) \parallel H(C(p))\big),$$

where $C(p)$ is the content of p. The hash value of a path p representing a directory is

$$\mathcal{H}(p) = H\big(\mathcal{H}(p_1) \parallel \mathcal{H}(p_2) \parallel \ldots \parallel \mathcal{H}(p_n) \parallel H(p)\big),$$

where $p_1, \ldots, p_n$ is a sorted list of all paths in p. We denote the root hash of a repository by $\mathcal{H}(\text{“.”})$.

It would be prohibitively expensive to recompute the hash values of all paths in a large repository upon every change of a single file. Therefore, the client stores the hash value of every path as an SVN property of the path. During an update operation, CSVN recomputes the hash values of all changed files and of all directories along the path from the changed files to the root. For a repository with n files, this reduces the cost of updating m modified files from linear in n to $\mathcal{O}(m + d)$, where d is the maximum affected depth in the directory tree.

The hash values are stored on the SVN server because properties are revision-controlled in SVN. Note that storing them on untrusted storage is unproblematic. The hash values are not actually needed by a client who checks out the complete repository because the client recomputes the entire hash tree anyway during verification. But they are needed for partial checkouts, as explained below.

Integration with SVN. During checkout and update operations, CSVN installs a callback before invoking the SVN library, which collects all relevant events reported by SVN; such relevant events are the addition, update, and deletion of a path. Then CSVN invokes CSS to obtain a revision number r and retrieves revision r from SVN, as described in Section 2.4. To recompute the root hash of the working copy, CSVN traverses the working copy, but visits only paths for which a relevant event was collected during the SVN operation.

For a commit operation, CSVN first determines the modified paths which are going to be written to the repository. It does that with an SVN "info" operation that outputs a collection of changed paths. Then it traverses the working copy, visiting and recomputing hash values only for changed paths, and getting hash values for unchanged paths from their SVN properties. This yields the root hash $h = \mathcal{H}(``.")$. CSVN further invokes the "commit" operation of SVN to write the updates to the repository and to obtain the new revision number r. Finally, it retrieves the revision commitment c from the last tuple in L, appends $(r, h, H(r \parallel h \parallel c))$ to L, and writes L using CSS.

This completes the description of the main CSVN operations. Further SVN operations can be implemented easily using the CSVN library and the three main CSVN operations.

The description so far assumes that clients always check out and update the complete file set in the repository at once. But this is not required in SVN, where a client may check out only a subdirectory from a repository, or commit only a subset of its working copy. The revision number and the root hash stored in CSS are always global properties of the repository, though. Operations on the partial repository are supported by our design and rely on the hash values stored in the SVN properties. For example, to check out a subtree from a repository, CSVN also needs to read all files along the path from the subtree's root to the repository root before it can verify the root hash.

An important and nice feature of this implementation is that it does not add any additional SVN server operations; because they usually involve the network and contain a cryptographic authentication operation during login, they tend to be rather slow.

4 Evaluation

We report on benchmarks to measure the performance of CSVN client operations in comparison to an unmodified SVN client. Since every operation of CSVN also invokes the corresponding operation of SVN, we are primarily interested in the overhead of CSVN over SVN.

We report on two kinds of performance evaluations: an application benchmark using real-life file sets of different sizes and a synthetic benchmark with artificially made-up file sets. Each benchmark consists of a series of tests executed by two clients, called A and B, where each test uses different data. For each test, we run the unmodified SVN client and the CSVN client 20 times in succession and measure the average time taken by each step in the test. Every run starts with an empty repository and a freshly initialized CSS. Each test uses a pair of related file sets; we are interested in the time it takes to update a working copy and the repository from one file set to the other one.

Each run in a test consists of the following steps:

1. Client A initializes a new empty repository on the server. This step is the same for both systems, so we do not measure it.
2. *Create* — client A checks out revision 0, creating a working copy from the empty repository.
3. *Import* — client A copies the first file set into its working copy, adds it to the repository, and commits the changes; we measure the time for the commit operation only.
4. *Checkout (CO) all* — client B checks out the content of the repository into its own working copy; the working copies of A and B are now identical.
5. Client B modifies its working copy to reflect the second file set. This involves adding the files contained only in the second file set, deleting the files only present in the first set, and copying the changed files from the second set into the working copy. This step is identical for both systems and is not measured.
6. *Commit (CI) diff* — client B commits the changes in its working copy.
7. *Update (UP) diff* — client A, whose working copy still contains the first file set, updates it to the most recent revision in the repository, which contains the second file set; the working copies of A and B are again identical.

This sequence of steps is designed to capture the overhead of committing and updating a large file set at once (in the *import* and *checkout all* steps) and of committing and updating smaller number of files in a larger file set (in the *commit diff* and *update diff* steps).

The benchmarks use two separate hosts, one for the server and one for both clients; they are connected by a gigabit LAN. The machine for the clients is an IBM x345 system with 2 GiB of RAM and two hyper-threaded Intel Xeon CPUs (3.06 GHz clock speed). The machine for the server is an IBM x335 system with 2 GiB of RAM and two hyper-threaded Intel Xeon CPUs (2.80 GHz clock speed). Both machines have a single IBM Ultra320 SCSI disk with 73.4 GB capacity and run Debian GNU/Linux 4.0 with kernel 2.6.18 and Subversion 1.5.2. The SVN server is accessed using SSH and all data is stored on the local filesystems. We use the SHA-1 hash function and 1024-bit RSA for signatures.

4.1 Application Benchmark

The file sets in our application benchmark are different versions of the Linux kernel source tree, as reported in Table 1. All files can be downloaded from the Linux kernel archive (`http://kernel.org/`). We choose them since they represent a realistic directory structure and because the repository sizes range over several orders of magnitude, from 632 KiB to 62 MiB. We selected the four versions that make up the first file set in a test based on their relative size. For each test, we pick the subsequently released version of the Linux kernel and use it as the second file set. The results are shown in Table 2.

Table 1. The four tests of the application benchmark and the used Linux kernel version pairs. The third and fourth columns list the number of files in the first file set and the number of changed (added, modified, or deleted) files between the two file sets, respectively.

Test (version pair)	Size	Files	Changed
0.11 → 0.12	0.63 MiB	100	91
1.0 → 1.0.1	5.9 MiB	561	12
2.0.1 → 2.0.2	27 MiB	2021	28
2.2.0 → 2.2.1	62 MiB	4599	10

Table 2. Results of the application benchmark. The numbers denote average elapsed time and standard deviation in seconds for SVN and CSVN in 20 runs, and the ratio of the two average times.

Step	SVN	CSVN	Ratio
Create	1.18 ±0.09	1.53 ±0.33	1.30
Import	0.93 ±0.02	1.94 ±0.00	2.08
CO all	0.99 ±0.00	1.10 ±0.00	1.11
CI diff	1.50 ±0.02	2.30 ±0.29	1.53
UP diff	0.94 ±0.00	1.05 ±0.01	1.11

Test 0.11 → 0.12

Step	SVN	CSVN	Ratio
Create	1.05 ±0.05	1.45 ±0.38	1.38
Import	3.70 ±0.11	6.76 ±0.00	1.83
CO all	1.98 ±0.00	3.17 ±0.48	1.60
CI diff	1.24 ±0.42	2.03 ±0.01	1.63
UP diff	0.94 ±0.01	1.11 ±0.00	1.17

Test 1.0 → 1.0.1

Step	SVN	CSVN	Ratio
Create	1.46 ±0.17	1.34 ±0.25	0.92
Import	14.08 ±0.71	28.84 ±1.02	2.05
CO all	7.49 ±2.38	12.31 ±2.44	1.64
CI diff	3.89 ±1.59	5.15 ±0.47	1.32
UP diff	0.94 ±0.00	2.28 ±0.02	2.42

Test 2.0.1 → 2.0.2

Step	SVN	CSVN	Ratio
Create	0.68 ±0.06	1.18 ±0.29	1.72
Import	36.35 ±1.05	79.26 ±1.29	2.18
CO all	13.38 ±2.04	29.28 ±2.28	2.19
CI diff	10.20 ±3.68	9.64 ±2.53	0.95
UP diff	1.27 ±1.53	1.69 ±0.49	1.33

Test 2.2.0 → 2.2.1

4.2 Synthetic Benchmark

In this benchmark, we wish to measure how the running time changes when we grow the directory structure in a repository from one directory to a large tree, but keep the number of files constant. To do this, we create four artificial file sets, each consisting of 256 files, each file of size 10 KiB, for a total data size of 2.5 MiB per file set. The files are filled with random pieces of C code taken from the Linux 2.2.1 kernel; this is to generate files looking like a real source tree. The files are stored in a directory structure of varying depth. We define a directory structure of depth d as a full binary tree of depth d and store $256/2^d$ files in each of the 2^d leaf directories.

Our file sets are four directory structures with depths 0 (all files in one directory), 2, 4, and 8 (every file in a separate directory). In each test, the second file

Table 3. Results of the synthetic benchmark. The numbers denote average elapsed time and standard deviation in seconds for SVN and CSVN in 20 runs, and the ratio of the two average times.

Step	SVN	CSVN	Ratio
Create	1.48 ±0.12	1.35 ±0.40	0.91
Import	2.71 ±0.06	4.19 ±0.50	1.55
CO all	1.99 ±0.00	2.90 ±0.01	1.46
CI diff	1.87 ±0.22	3.02 ±0.01	1.61
UP diff	0.95 ±0.00	1.73 ±0.01	1.83

Depth 0

Step	SVN	CSVN	Ratio
Create	1.25 ±0.06	1.25 ±0.39	1.00
Import	2.01 ±0.33	3.89 ±0.01	1.93
CO all	1.99 ±0.00	2.40 ±0.01	1.21
CI diff	0.93 ±0.05	1.51 ±0.01	1.63
UP diff	0.95 ±0.00	1.01 ±0.01	1.07

Depth 2

Step	SVN	CSVN	Ratio
Create	1.27 ±0.01	1.46 ±0.39	1.15
Import	1.95 ±0.27	3.88 ±0.01	1.99
CO all	1.99 ±0.01	2.30 ±0.02	1.16
CI diff	0.92 ±0.06	1.61 ±0.03	1.75
UP diff	0.95 ±0.00	0.94 ±0.01	1.00

Depth 4

Step	SVN	CSVN	Ratio
Create	0.86 ±0.23	1.33 ±0.14	1.55
Import	4.33 ±0.42	10.59 ±0.89	2.44
CO all	8.75 ±1.52	9.95 ±1.14	1.14
CI diff	0.88 ±0.04	1.31 ±0.24	1.50
UP diff	2.28 ±1.15	1.90 ±0.51	0.84

Depth 8

set is identical to the first one, up to a random modification to one of the files in a leaf directory. The results are shown in Table 3.

4.3 Results

The results of both benchmarks show that CSVN adds an overhead of a factor that is generally less than 2 and usually also less than 1.5. In absolute terms, the *import* and the *checkout all* steps are the slowest operations because they involve all files. The *import* step generally incurs also the biggest overhead, usually around 2. But the overhead of the *checkout all* step is not noticeably different from the overhead of the remaining steps. Generally, CSVN adds only a moderate overhead to most operations compared to the normal SVN client.

Observe the bigger variation in the execution times of the tests with larger file sets. One reason for this effect may be that large data sets create more unexpected interactions with other programs due to swapping and disk operations than small data sets that fit in the kernel's buffer cache. Such variations also explain the few overhead ratios smaller than 1.

Among the results of the application benchmark in Table 2, the second largest overhead (after the *import* step) usually occurs for the *commit diff* step. The overhead on the large file sets is not bigger than that on the smaller file sets. This clearly shows the benefit of using a hash tree when only a small part of a large file set is updated.

In the results of the synthetic benchmark in Table 3, observe the overhead of the *commit diff* and the *update diff* steps. In both steps, only a single file is changed. The CSVN client must then read the hash values of all sibling files to compute the new hash values for the directory. With the increasing depth of the directory structure, the number of sibling files drops from 255 to 0, and this is reflected in the decreasing overhead.

In summary, although a 50%–100% larger execution time for SVN operations is clearly noticeable by the clients, we believe it is a reasonable price to pay for the added guarantee of cryptographically verified data integrity. These results should serve as a lower bound for the efficiency of our design, because they were carried out with our straight-forward layered prototype implementation in Python. If the CSVN operations would be integrated with the SVN client library, the directory tree in the working would have to be traversed only once instead of twice; moreover, hashing could be integrated with the traversal and performed concurrently with receiving or sending data to the server. With such an integrated design, the cryptographic overhead is likely to vanish, as shown in other benchmarks of cryptographic storage and file systems [19].

5 Conclusions

Protecting data integrity against unauthorized modifications is an important aspect of networked storage systems. This paper presented a novel approach to securing the integrity of data stored in revision control systems, and demonstrated its feasibility with our Consistent Subversion (CSVN) prototype. Our evaluation shows that the overhead is reasonable.

The biggest threat to our system are client failures. Protecting the system from malicious clients is also the area where future work is needed.

Our implementation already tolerates client crashes; one or more malicious clients alone cannot harm the integrity *if* the service provider is correct — measures to prevent such behavior can easily be added [11], but have not been described in this work. A corrupted client conspiring with a corrupted service provider, however, may undermine fork-linearizability.

A first barrier against such an attack is the CA that must authorize all clients before they access the service. It is therefore a good idea to make the CA is a separate entity from the storage service. If the threat of such a client-server conspiracy attack becomes too serious, one might adopt the complex cross-checking of versions signed by different clients introduced in SUNDR [14]. Unfortunately, the SUNDR protocol involves a much higher communication overhead in every operation. One should also develop an additional tool that helps the clients to recover from a server failure; it should automatically reconcile the state of the repository from the information held by the clients in their working copies and their local memories.

Acknowledgments

We are grateful to Idit Keidar, Alexander Shraer, and Marko Vukolić for many discussions and valuable comments.

This work was supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

References

[1] Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic snapshots of shared memory. Journal of the ACM 40(4), 873–890 (1993)

[2] Baker, M., Shah, M., Rosenthal, D.S.H., Roussopoulos, M., Maniatis, P., Giuli, T., Bungale, P.: A fresh look at the reliability of long-term digital storage. In: Proc. 1st European Conference on Computer Systems (EuroSys), pp. 221–234 (2006)

[3] Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. Algorithmica 12, 225–244 (1994)

[4] Cachin, C., Keidar, I., Shraer, A.: Fail-aware untrusted storage. In: Proc. International Conference on Dependable Systems and Networks (DSN-DCCS) (2009)

[5] Cachin, C., Shelat, A., Shraer, A.: Efficient fork-linearizable access to untrusted shared memory. In: Proc. 26th ACM Symposium on Principles of Distributed Computing (PODC), pp. 129–138 (August 2007)

[6] Clarke, D., Suh, G.E., Gassend, B., Sudan, A., van Dijk, M., Devadas, S.: Towards constant bandwidth overhead integrity checking of untrusted data. In: Proc. 26th IEEE Symposium on Security & Privacy (2005)

[7] CNET News. Red Hat, Fedora servers compromised (August 2008), `http://news.cnet.com/8301-1009_3-10023565-83.html`

[8] Goh, E.-J., Shacham, H., Modadugu, N., Boneh, D.: SiRiUS: Securing remote untrusted storage. In: Proc. Network and Distributed Systems Security Symposium (NDSS) (2003)

[9] Haber, S., Stornetta, W.S.: How to time-stamp a digital document. Journal of Cryptology 3, 99–111 (1991)

[10] Herlihy, M., Shavit, N.: The Art of Multiprocessor Programming. Morgan Kaufmann, San Francisco (2008)

[11] Li, J., Krohn, M., Maziéres, D., Shasha, D.: Secure untrusted data repository (SUNDR). In: Proc. 6th Symp. Operating Systems Design and Implementation (OSDI), pp. 121–136 (2004)

[12] Maheshwari, U., Vingralek, R., Shapiro, W.: How to build a trusted database system on untrusted storage. In: Proc. 4th Symp. Operating Systems Design and Implementation (OSDI) (2000)

[13] Mazières, D., Kaminsky, M., Kaashoek, F., Witchel, E.: Separating key management from file system security. In: Proc. 17th ACM Symposium on Operating System Principles (SOSP) (1999)

[14] Mazières, D., Shasha, D.: Building secure file systems out of Byzantine storage. In: Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC) (2002)

[15] Merkle, R.C.: Protocols for public-key cryptosystems. In: Proc. IEEE Symposium on Security & Privacy, pp. 122–133 (1980)

[16] Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. ACM Transactions on Storage 2(2), 107–138 (2006)

[17] Papamanthou, C., Tamassia, R., Triandopoulos, N.: Authenticated hash tables. In: Proc. 15th ACM Conference on Computer and Communications Security (2008)

[18] Siong, N.P., Toivonen, H.: M2Crypto Python interface to OpenSSL. Version 0.18.2 (2008), `http://chandlerproject.org/Projects/MeTooCrypto`

[19] Wright, C.P., Dave, J., Zadok, E.: Cryptographic file systems performance: What you don't know can hurt you. In: Proc. 2nd International IEEE Security in Storage Workshop (SISW) (2003)

Fragility of the Robust Security Network: 802.11 Denial of Service

Martin Eian

Department of Telematics
Norwegian University of Science and Technology
martin.eian@item.ntnu.no

Abstract. The upcoming 802.11w amendment to the 802.11 standard eliminates the 802.11 deauthentication and disassociation Denial of Service (DoS) vulnerabilities. This paper presents two other DoS vulnerabilities: one vulnerability in draft 802.11w implementations discovered by IEEE 802.11 TGw, and one new vulnerability in 802.11, which is still present in the 802.11w amendment. Attacks exploiting the first vulnerability are significantly more efficient than any known 802.11 DoS attacks, while attacks exploiting the second vulnerability have efficiency and feasability equivalent to a disassociation attack. This paper provides an experimental verification of these attacks, demonstrating their feasability using freely available software and off the shelf hardware. Finally, the root cause of these vulnerabilities is discussed and a backwards compatible solution proposed.

Keywords: Wireless, Security, Denial of Service, 802.11, 802.11i, 802.11w.

1 Introduction

In the original IEEE 802.11 standard[9], ratified in 1997 and accepted as an ISO standard in 1999, the only available security mechanism was Wired Equivalent Privacy (WEP). During the years that followed, WEP was analyzed by the academic community and wireless hackers, and several vulnerabilities were discovered [8] [15] [5]. This motivated the development of a replacement for WEP, IEEE 802.11i. In 2004, the 802.11i amendment was ratified, with two new and improved security mechanisms. The first one, Temporal Key Integrity Protocol (TKIP), was designed as a transitional solution that would support old hardware. The second, counter mode with cipher-block chaining message authentication code protocol (CCMP), was the long term solution to the security vulnerabilities of WEP. The common denominator for WEP, TKIP and CCMP is that they protect 802.11 data frames. No protection is provided for control frames and management frames.

One issue with the lack of management frame protection is that any station on the wireless network can transmit forged management frames. This tactic can be used by an attacker to make a station (STA) deauthenticate or disassociate from the access point (AP). The following association request from the

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 400–416, 2009.

station gives the attacker the service set identifier (SSID) of the wireless network, thus bypassing SSID cloaking. Furthermore, dictionary attacks against TKIP or CCMP using a password derived preshared key (PSK) require that the attacker observes the initial 4-way handshake, and a successful disassociation attack will result in this 4-way handshake between the wireless station and the AP. Last, but not least, transmitting deauthentication or disassociation frames several times per second is a very efficient Denial of Service attack on the wireless network. Aireplay-ng from the aircrack-ng[1] suite is an example of a freely available tool that implements the deauthentication attack. One countermeasure to these attacks is to provide integrity and replay protection for management frames.

Another issue that has surfaced recently is that several of the new amendments to the 802.11 standard extend the use of management action frames, transmitting potentially sensitive information inside management frames. Examples of such amendments are 802.11k, 802.11r and 802.11v. To avoid the compromise of sensitive information, management frame confidentiality must be provided.

As a response to the above mentioned issues, Task Group w (TGw) was established in 2005 to develop the 802.11w amendment, Protected Management Frames. The original target date for ratification of this amendment was September 2007, but this was later postponed to December 2009. The design goal for 802.11w was to extend the security mechanisms in 802.11i to provide protection for selected 802.11 management frames. 802.11w is currently in draft status. The newest available draft version is 7.0.

The results presented in this paper are based on IEEE 802.11-2007[13], which includes the 802.11i amendment[10], and 802.11w draft version 3.0[12] from September 2007. One additional feature from 802.11w draft version 4.0, protection against SA termination attacks, is also discussed. The analysis of potential DoS vulnerabilities in 802.11 with amendments is based on the observations in [14].

The rest of the paper is divided into eight sections. Section 2 presents the contribution. In Section 3, a short description of related work on 802.11 DoS vulnerabilities is presented. Section 4 contains an analysis of relevant topics from the 802.11 standard with amendments. Section 5 presents theoretical DoS vulnerabilities in 802.11, 802.11i and 802.11w and some general observations on network DoS. Section 6 provides a description of the experiments, analysis and results. The results are discussed in Section 7, and a solution proposed in Section 8. Section 9 contains the conclusion and section 10 contains acknowledgements.

2 Contribution

This paper analyzes medium access control (MAC) layer DoS vulnerabilities in 802.11 with the 802.11i and 802.11w amendments. One apology for MAC layer DoS vulnerabilities is that an attacker can use physical jamming of the radio frequencies to perform a DoS attack anyway, which is extremely difficult to prevent. The motivation for preventing DoS attacks against the MAC layer is that such attacks are far more efficient than jamming, so the attacker has to spend less effort, and thus will be more difficult to detect and locate. Furthermore, certain attacks against MAC layer vulnerabilities may cause a deadlock such that a station is not able to recover. A jamming attack, on the other hand, will only disrupt network access for as long as the attacker is transmitting.

The configuration used for the experimental analysis is an extended service set (ESS) with a wireless station communicating with an AP. The term station refers to either a non-AP 802.11 device or an AP.

This paper makes three principal contributions. First, a previously unknown DoS vulnerability in 802.11, equivalent to the disassociation vulnerability, and still present in 802.11w, is presented and analyzed. Second, this new vulnerability is tested experimentally together with the deauthentication attack and another vulnerability discovered by J. Epstein in 2007[6]. All experiments were carried out using freely available tools and off the shelf hardware. Third, a robust solution to the MAC layer DoS vulnerabilities in 802.11 is proposed. It is possible to introduce this solution incrementally, preserving backwards compatibility until all APs and stations are upgraded.

3 Related Work

In 2003, Bellardo and Savage demonstrated the feasability and efficiency of the 802.11 deauthentication attack, together with several other DoS attacks against the 802.11 MAC layer[4]. [4] is a useful general reference on DoS attacks against 802.11 networks. In 2007, J. Epstein presented the theoretical SA termination attack[6] and a proposed solution[7] to TGw, which was accepted as part of draft 4.0 of the 802.11w amendment in 2008. The SA termination attack and the proposed solution are analyzed in this paper. The working documents of TGw are available at `https://mentor.ieee.org/802.11/documents`.

4 Analysis of the 802.11 Standard

Only the most relevant parts of the 802.11 standard and the 802.11i and 802.11w amendments are presented as background material. The reader is referred to the IEEE standard and draft documents for a comprehensive review.

4.1 802.11 Authentication and Association

The original 802.11 standard specifies two types of authentication: shared key and open system. The shared key authentication is optional in WEP, and the open system authentication is a two-message null authentication initiated by the station. After authentication, the station performs an association with the AP. Figure 1 shows a successful open system authentication followed by a successful association.

Associations are used to keep track of the stations served by an AP. The 802.11 standard defines two state variables: authentication state and association

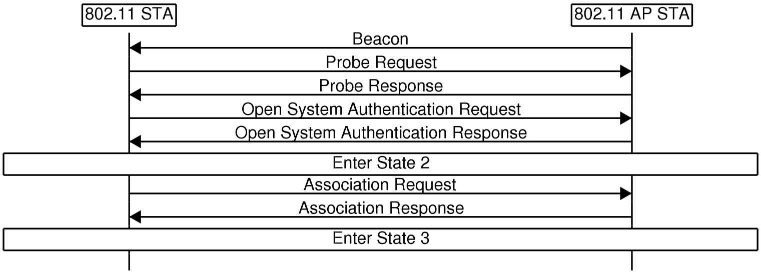


Fig. 1. 802.11 open system authentication and association

Table 1. 802.11 States

State 1	Not authenticated	Not associated
State 2	Authenticated	Not associated
State 3	Authenticated	Associated

state. Three of the four possible combinations of these two variables represent the local 802.11 station states shown in Table 1. Every station maintains a local state for every other station that it communicates with.

802.11 frames are grouped into classes that correspond to the states mentioned above. Frames corresponding to the current state or lower are allowed, thus the allowed frames in State 2 are of Class 1 or 2. If a station receives a Class 2 or 3 frame from a station that is not authenticated, it shall respond with a deauthentication frame. If it receives a Class 3 frame from a station that is authenticated, but not associated, it shall respond with a disassociation frame. Figure 2 shows the valid transitions between the local states in 802.11.

Subsection 11.3.1.2 of the 802.11 standard[13] specifies how the destination STA should handle 802.11 authentication requests:

> Upon receipt of an Authentication frame with authentication transaction sequence number equal to 1, the destination STA shall authenticate with the indicated STA using the following procedure:
>
> a) The STA shall execute the authentication mechanism described in 8.2.2.2.
>
> b) If the authentication was successful, the state variable for the indicated STA shall be set to State 2.
>
> c) The STA shall issue an MLME-AUTHENTICATE.indication primitive to inform the SME of the authentication.

Note that an open system authentication will always be successful, so an AP that receives an open system authentication request will always enter State 2 (authenticated, but not associated).

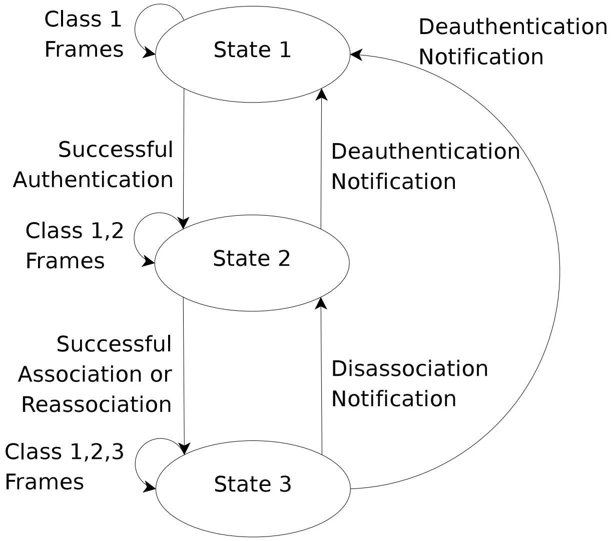


Fig. 2. 802.11 state transitions. The authentication attack triggers a change from State 3 to State 2 in the AP by transmitting a forged open system authentication request. This transition is not shown in the state diagram, the only transition from State 2 to State 3 is a disassociation notification, but it must be allowed to avoid deadlocks when 802.11w is enabled.

4.2 802.11i Security Amendments

802.11i introduces a new security framework: The Robust Security Network (RSN). Authentication and key management in an RSN is carried out after the successful completion of 802.11 authentication and association, as illustrated in figure 1. However, some of the messages are modified. The beacon frames broadcast by the AP and the probe response contain an RSN information element with the supported security parameters. Cryptographic parameters are negotiated during the association phase by including an RSN information element in the association request from the station. If the security parameters are accepted by the AP, it enters State 3, and authentication is carried out using the Extensible Authentication Protocol (EAP)[3]. EAP encapsulation over Local Area Networks (EAPOL), as specified in IEEE 802.1X[11], is used to encapsulate the authentication messages in 802.11 data frames. Figure 3 shows the authentication and key management in an RSN.

802.11i uses security associations (SAs) to store security policies and cryptographic keys. There are two parts of the SA specifications that are relevant to the vulnerabilities discussed in this paper: SA termination and recovery from lost key state synchronization.

SA termination is triggered when an AP receives or transmits certain management frames. If an AP receives a valid association or reassociation frame from a station, it will delete the pairwise transient key SA (PTKSA), which contains

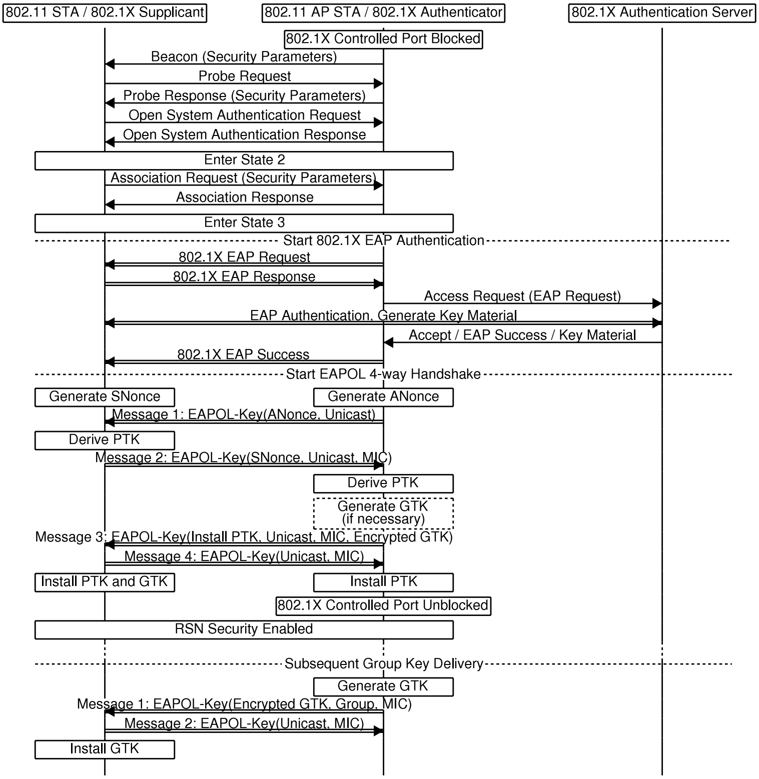

802.11 STA / 802.1X Supplicant
802.11 AP STA / 802.1X Authenticator
802.1X Authentication Server
802.1X Controlled Port Blocked
Beacon (Security Parameters)
Probe Request
Probe Response (Security Parameters)
Open System Authentication Request
Open System Authentication Response
Enter State 2
Association Request (Security Parameters)
Association Response
Enter State 3
Start 802.1X EAP Authentication
802.1X EAP Request
802.1X EAP Response
Access Request (EAP Request)
EAP Authentication, Generate Key Material
Accept / EAP Success / Key Material
802.1X EAP Success
Start EAPOL 4-way Handshake
Generate SNonce
Generate ANonce
Message 1: EAPOL-Key(ANonce, Unicast)
Derive PTK
Message 2: EAPOL-Key(SNonce, Unicast, MIC)
Derive PTK
Generate GTK
(if necessary)
Message 3: EAPOL-Key(Install PTK, Unicast, MIC, Encrypted GTK)
Message 4: EAPOL-Key(Unicast, MIC)
Install PTK and GTK
Install PTK
802.1X Controlled Port Unblocked
RSN Security Enabled
Subsequent Group Key Delivery
Generate GTK
Message 1: EAPOL-Key(Encrypted GTK, Group, MIC)
Message 2: EAPOL-Key(Unicast, MIC)
Install GTK

Fig. 3. 802.11i RSN authentication and key management. Single lines represent management frames, double lines represent data frames. Note that a deauthentication attack will force the station to do the whole procedure over again, starting with the authentication request. If pairwise master key security association (PMKSA) caching is enabled, the 802.1X authentication does not have to be repeated.

the station's pairwise transient key. The PTKSA is also deleted if the AP sends or receives a deauthentication or disassociation frame.

Loss of key state synchronization can occur if a station reboots and the temporal keys stored in memory are lost. A station that loses key state synchronization in an ESS shall perform the deauthentication procedure before it sends an authentication request. If the authentication and key management protocol (AKMP) fails between a station and an AP that are associated, both the station and AP shall perform the deauthentication procedure.

4.3 802.11w Protected Management Frames

802.11w uses CCMP from 802.11i to provide integrity, confidentiality and sender authenticity for unicast management frames, and Broadcast Integrity Protocol (BIP) to provide integrity for broadcast management frames. In both cases, protection is only provided for management frames of subtype action, deauthentication and disassociation. If protection of management frames is enabled and an unprotected management frame of subtype action, deauthentication or disassociation is received, the frame is silently discarded.

5 Vulnerability Analysis

5.1 General Observations

Meadows discusses several important principles for protocol design to minimize the vulnerability to DoS attacks[14]. One of the fundamental principles is the following:

> First of all, such a protocol must provide authentication from the very beginning.

802.11 with the 802.11i and 802.11w amendments does not provide this, since the 802.11 authentication and association procedures are carried out, unprotected, before the 802.11i authentication is initiated. All of the messages exchanged prior to the 802.11i authentication can thus be forged by an attacker. Of particular interest are the messages that result in state transitions for the AP: authentication requests, association requests, deauthentication notifications and disassociation notifications. A successful authentication request will make the AP enter State 2. A successful association request will make the AP enter State 3 if it is currently in State 2. Deauthentication and disassociation notifications will make the AP enter State 1 or State 2, respectively. The 802.11w amendment provides integrity protection for deauthentication and disassociation notifications, and in the latest drafts it also provides a mechanism to avoid forged association requests. Authentication requests, however, are not protected. Exploiting unprotected authentication requests to perform a DoS attack against 802.11 with 802.11i and 802.11w is a principal contribution of this paper.

5.2 The 802.11 Standard

802.11 deauthentication and disassociation DoS attacks are carried out by forging a deauthentication or disassociation frame. The receiving station will change to State 1 for a deauthentication or State 2 for a disassociation. The most efficient of these two is the deauthentication attack. If the station is deauthenticated, it has to authenticate and associate to be able to send and receive traffic again. A slightly more efficient approach is to deauthenticate the AP, which resets the AP to State 1. The next data frame from the station will be dropped, the AP will respond with a deauthentication notification, and the station will then authenticate and associate.

5.3 802.11i Security Amendments

802.11i significantly "improves" the efficiency of the deauthentication DoS attack. Once a station has been deauthenticated, it must first perform 802.11 authentication and association. Then, if enabled, 802.1X authentication must be carried out. 802.1X authentication is not used with TKIP-PSK and CCMP-PSK, or when PMK caching is enabled and a valid PMKSA exists between the AP and station. Finally, an EAPOL 4-way handshake must be completed to derive the temporal keys. Once the 4-way handshake is completed, the station can send and receive traffic.

The SA termination procedures in 802.11i make an even more efficient DoS attack possible. If an attacker sends a forged association or reassociation frame from the station to the AP, the AP will remain in State 3, but the temporal keys will be deleted. The AP will start the EAPOL 4-way handshake, which will eventually time out, then deauthenticate the station, resulting in the procedure described in the previous paragraph.

5.4 802.11w Protected Management Frames

802.11w prevents the deauthentication and disassociation attacks. However, the effect of the SA termination attack is amplified. When the EAPOL 4-way handshake times out, the AP will try to deauthenticate the station. Since the pairwise keys in the AP are deleted, the deauthentication frame will not be protected, and thus discarded by the station. The station will not be able to send or receive any traffic, and is not able to recover, since it discards the deauthentication frames from the AP. An attempt to fix this vulnerability is included in draft version 4.0 and later of 802.11w, where a cryptographically protected SA Query procedure is used to determine whether or not an association or reassociation frame from the station is legitimate. Implementations based on draft 3.0 or earlier, however, are still vulnerable to the SA termination attack.

The SA Query procedure works as follows: if an AP receives an association request from a station with which it has a valid PTKSA, the AP responds that the association request was temporarily rejected. This response tells the station how long it has to wait before it can send another association request. Then, the AP tries to send one or more query messages to the station to check if it has a valid PTKSA. The queries are management action frames protected under the current PTKSA. If a valid response to one of these queries is received, the association request is ignored. If no response is received before the timeout value is reached, the AP will delete the PTKSA. A station that loses key state synchronization will thus have to send an association request, wait until the query procedure times out, then send a new association request. The number of queries and timeout value are configurable parameters.

Another issue with 802.11w is that the recovery procedure for lost key state synchronization in 802.11i is no longer possible, since a station that loses synchronization will not be able to send a protected deauthentication frame to the AP. To recover, a station has to start 802.11 authentication without first performing a deauthentication, and the AP has to allow this to avoid a deadlock.

This can be exploited to enable a new kind of DoS attack against 802.11: The attacker transmits a forged open system authentication frame, which will make the AP enter State 2. The AP still has a valid PTKSA with the station, so once the station transmits a data frame, the AP responds with a protected disassociation frame. The end result is the same as if a disassociation attack had been carried out. This type of attack will from now on be referred to as an “authentication attack”.

6 Experiments

The goals of the experiments were to verify the feasability of the authentication and SA termination DoS attacks, and to verify that 802.11w protects against the deauthentication attack. To this end, the authentication, SA termination and deauthentication attacks were performed both with 802.11w enabled and disabled, for a total of six experiments. Each attack was performed 100 times to ensure that the results were consistent.

6.1 Infrastructure Set-Up

The infrastructure under attack consisted of a Cisco 4402 wireless controller (AIR-WLC4402-25-K9) and a Cisco 1030 access point (AIR-AP-1030). Both the wireless controller and access point were running software version 4.2.61.0 with Cisco Management Frame Protection (MFP) based on an earlier 802.11w revision than draft 3.0. 802.11i CCMP-PSK was used for all the experiments. The wireless controller and AP were configured to reject shared key authentication, and CCMP-PSK was required, which means that association requests without an RSN information element were rejected. The station was a laptop computer with a Cisco Aironet 802.11 a/b/g network adapter (AIR-CB21AG-E-K9), running Windows XP SP2. The station was assigned an IPv4 address through DHCP from the wireless controller. Both the AP and station used 802.11g for the experiments. The attacker was a laptop with a wireless network interface card (NIC) with the Atheros AR2413 chipset, running Linux 2.6.22 with the madwifi-ng drivers, and aircrack-ng[1] version 0.9.1 as the attack software. In particular, airmon-ng was used to enable RFMON (monitor) mode, aireplay-ng and airtun-ng were used to inject frames, and airodump-ng to capture traffic. The same wireless network interface was used for frame injection and traffic capture, and the experiments were conducted in a typical office environment, with no shielding from other wireless stations and APs nearby. The only legitimate traffic on the wireless network was an Internet Control Message Protocol (ICMP) ECHO request from the station to a server on the wired LAN every second, and an ICMP ECHO request from the server to the station every second, along with the ICMP ECHO responses. The ping commands in Windows XP and Linux were used to generate traffic from the station and server, respectively.

6.2 Attacks

The attacks were carried out by transmitting a single management frame of subtype authentication request, association request or deauthentication. To ensure that only one frame was transmitted, an attack tool was used to generate the frame, which was captured using airodump-ng. The single frame was then saved to a file and replayed using airtun-ng.

First, the aireplay-ng tool was used to generate an authentication request frame, with the two-byte authentication algorithm field set to "Open System" ($0x0000$). Then, an association request frame containing an RSN information element with CCMP support, which would be accepted by the AP, was obtained by running an authentication attack and recording the subsequent association request transmitted by the station. It is also possible for an attacker to construct a valid association frame from the information contained in the beacon frames broadcast from the AP. The authentication request and association request frames were constructed with the station MAC address as source and the AP MAC address as destination. Last, the aireplay-ng tool was used to construct a deauthentication frame with the AP MAC address as source and the station MAC address as destination.

Once the attack frames were generated, the experiments were performed by transmitting an attack frame once, then waiting for the station to regain connectivity. Once the station was back on-line, a new attack was launched, and this was repeated until a total of 100 attacks of each type had been carried out. All 802.11 frames to and from the AP were recorded for analysis.

6.3 Observations

Several significant results were observed while conducting the experiments.

First, as expected, the deauthentication attack did not work when MFP was enabled, but did work as expected when disabled.

Second, the authentication attack worked, both with and without MFP enabled. The station lost its network connection and had to reconnect.

Third, the valid association attack resulted in a permanent DoS when MFP was enabled. After excessive timeouts, the station interface was automatically assigned a link-local IPv4 address (169.254/16 prefix), and manual intervention was needed to get it back on-line.

6.4 Results

Once the experiments were completed, Wireshark[2] version 0.99.6 was used to analyze the results.

Deauthentication Attack. The deauthentication attack worked as expected. Figure 4 shows the expected and observed results when MFP was disabled. With MFP enabled, the attack had no effect, since the deauthentication notification was ignored by the station.

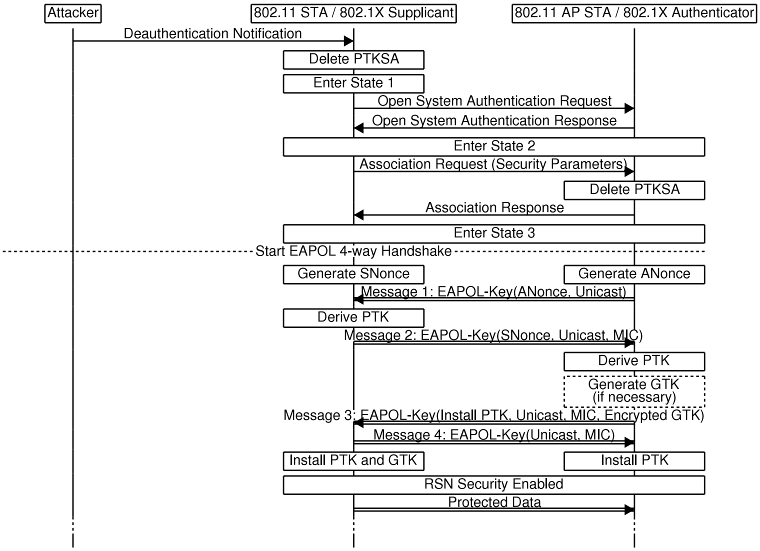

Attacker
802.11 STA / 802.1X Supplicant
802.11 AP STA / 802.1X Authenticator
Deauthentication Notification
Delete PTKSA
Enter State 1
Open System Authentication Request
Open System Authentication Response
Enter State 2
Association Request (Security Parameters)
Delete PTKSA
Association Response
Enter State 3
Start EAPOL 4-way Handshake
Generate SNonce
Generate ANonce
Message 1: EAPOL-Key(ANonce, Unicast)
Derive PTK
Message 2: EAPOL-Key(SNonce, Unicast, MIC)
Derive PTK
Generate GTK (if necessary)
Message 3: EAPOL-Key(Install PTK, Unicast, MIC, Encrypted GTK)
Message 4: EAPOL-Key(Unicast, MIC)
Install PTK and GTK
Install PTK
RSN Security Enabled
Protected Data

Fig. 4. Expected and observed results for the deauthentication attack with MFP disabled. The attack had no effect when MFP was enabled.

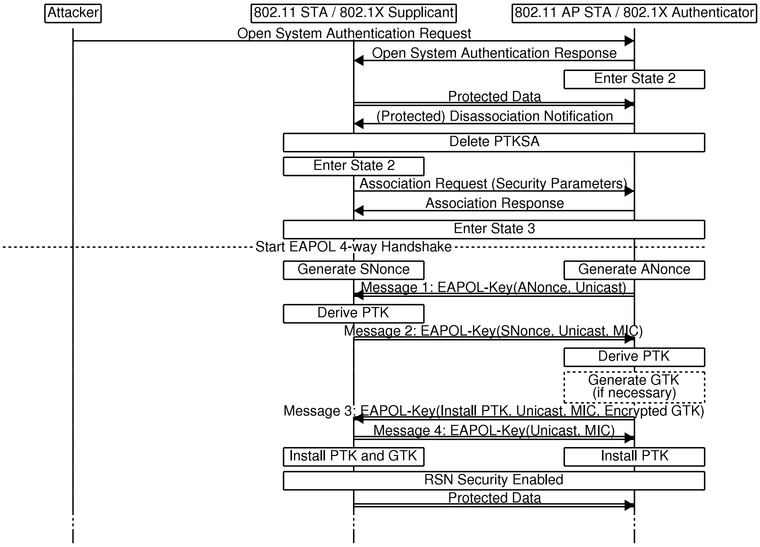

Attacker
802.11 STA / 802.1X Supplicant
802.11 AP STA / 802.1X Authenticator
Open System Authentication Request
Open System Authentication Response
Enter State 2
Protected Data
(Protected) Disassociation Notification
Delete PTKSA
Enter State 2
Association Request (Security Parameters)
Association Response
Enter State 3
Start EAPOL 4-way Handshake
Generate SNonce
Generate ANonce
Message 1: EAPOL-Key(ANonce, Unicast)
Derive PTK
Message 2: EAPOL-Key(SNonce, Unicast, MIC)
Derive PTK
Generate GTK (if necessary)
Message 3: EAPOL-Key(Install PTK, Unicast, MIC, Encrypted GTK)
Message 4: EAPOL-Key(Unicast, MIC)
Install PTK and GTK
Install PTK
RSN Security Enabled
Protected Data

Fig. 5. Expected results for the authentication attack. The only difference between MFP enabled and disabled was that in the former case the disassociation notification was protected.

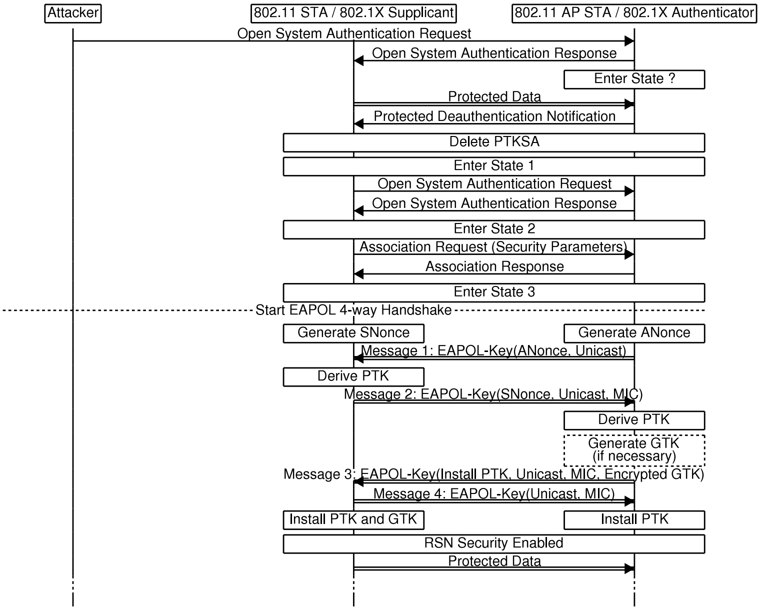

Attacker
802.11 STA / 802.1X Supplicant
802.11 AP STA / 802.1X Authenticator
Open System Authentication Request
Open System Authentication Response
Enter State ?
Protected Data
Protected Deauthentication Notification
Delete PTKSA
Enter State 1
Open System Authentication Request
Open System Authentication Response
Enter State 2
Association Request (Security Parameters)
Association Response
Enter State 3
Start EAPOL 4-way Handshake
Generate SNonce
Generate ANonce
Message 1: EAPOL-Key(ANonce, Unicast)
Derive PTK
Message 2: EAPOL-Key(SNonce, Unicast, MIC)
Derive PTK
Generate GTK (if necessary)
Message 3: EAPOL-Key(Install PTK, Unicast, MIC, Encrypted GTK)
Message 4: EAPOL-Key(Unicast, MIC)
Install PTK and GTK
Install PTK
RSN Security Enabled
Protected Data

Fig. 6. Observed results for the authentication attack with MFP enabled. The AP responds with a deauthentication notification when it should have used a disassociation notification.

Authentication Attack. The results of the authentication attack were slightly different from the expected results. Figure 5 shows how the attack would work on an implementation that conforms to the standard.

The only difference between the expected and observed results were that the AP responded with a deauthentication notification when it should have used a disassociation notification. Figure 6 shows the observed results with MFP enabled. With MFP disabled, the only difference was that the deauthentication notification was not protected. The reason code in the deauthentication notification frame was “Class 3 frame received from nonassociated station (0x0007)”, which confirms that the AP was in State 1 or 2 immediately after the attack.

SA Termination Attack. The results of the SA termination attacks were also as expected. Figure 7 shows the expected and observed results with MFP disabled. One interesting observation is that this attack is more efficient than any other known MAC layer DoS attack against 802.11 when RSN is enabled. In the experiment, the AP sent the first message of the EAPOL 4-way handshake, then waited for one second before retrying. This was repeated three times before the AKMP failed. The SA termination attach thus added three more seconds of downtime compared to the deauthentication attack.

Figure 8 shows the expected and observed results with MFP enabled. The station did not accept unprotected deauthentication notifications from the AP.

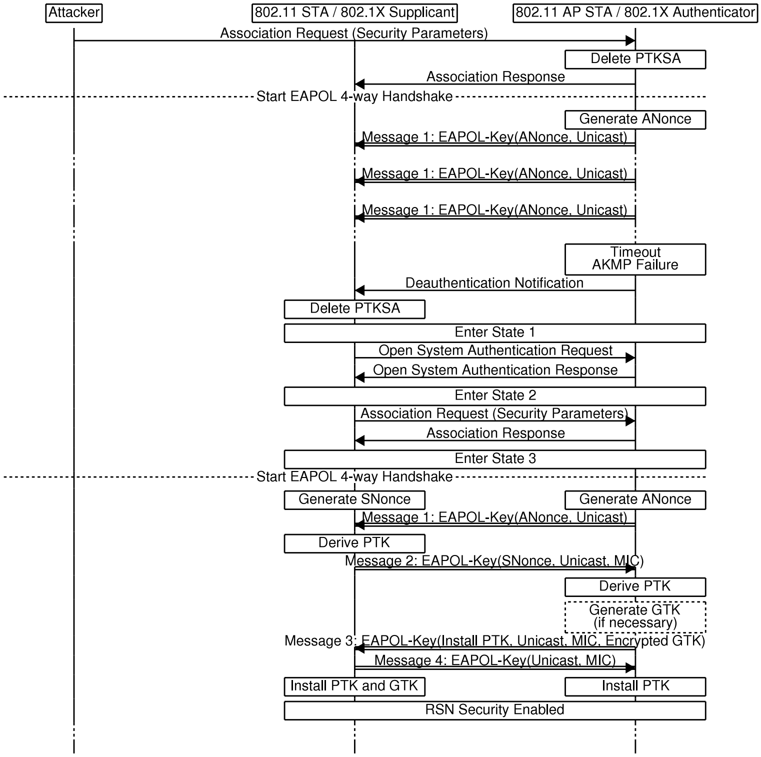

Attacker
802.11 STA / 802.1X Supplicant
802.11 AP STA / 802.1X Authenticator
Association Request (Security Parameters)
Delete PTKSA
Association Response
Start EAPOL 4-way Handshake
Generate ANonce
Message 1: EAPOL-Key(ANonce, Unicast)
Message 1: EAPOL-Key(ANonce, Unicast)
Message 1: EAPOL-Key(ANonce, Unicast)
Timeout
AKMP Failure
Deauthentication Notification
Delete PTKSA
Enter State 1
Open System Authentication Request
Open System Authentication Response
Enter State 2
Association Request (Security Parameters)
Association Response
Enter State 3
Start EAPOL 4-way Handshake
Generate SNonce
Generate ANonce
Message 1: EAPOL-Key(ANonce, Unicast)
Derive PTK
Message 2: EAPOL-Key(SNonce, Unicast, MIC)
Derive PTK
Generate GTK
(if necessary)
Message 3: EAPOL-Key(Install PTK, Unicast, MIC, Encrypted GTK)
Message 4: EAPOL-Key(Unicast, MIC)
Install PTK and GTK
Install PTK
RSN Security Enabled

Fig. 7. Expected and observed results for the SA termination attack with MFP disabled. The failed 4-way handshake adds three seconds of downtime compared to a deauthentication attack.

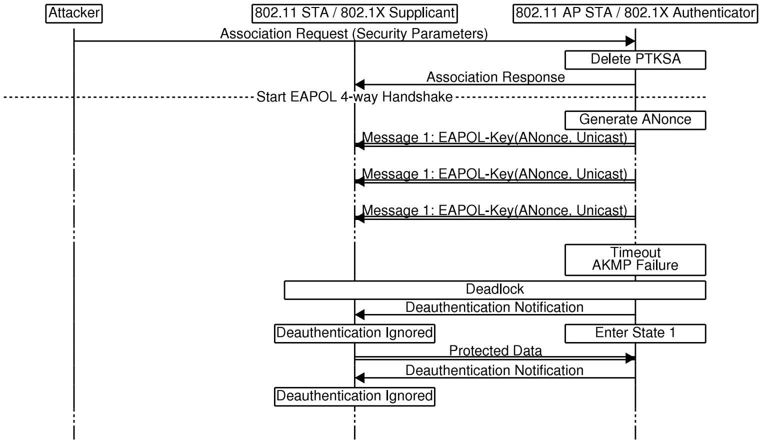

Attacker
802.11 STA / 802.1X Supplicant
802.11 AP STA / 802.1X Authenticator
Association Request (Security Parameters)
Delete PTKSA
Association Response
Start EAPOL 4-way Handshake
Generate ANonce
Message 1: EAPOL-Key(ANonce, Unicast)
Message 1: EAPOL-Key(ANonce, Unicast)
Message 1: EAPOL-Key(ANonce, Unicast)
Timeout
AKMP Failure
Deadlock
Deauthentication Notification
Deauthentication Ignored
Enter State 1
Protected Data
Deauthentication Notification
Deauthentication Ignored

Fig. 8. Expected and observed results for the SA termination attack with MFP enabled. The result is a deadlock.

The end result was a deadlock, with manual intervention required to get the station reconnected.

7 Discussion

The results from the theoretical analysis and the experiments in the previous sections show that a network using 802.11w is vulnerable to the authentication attack. This attack has the same efficiency and feasability as a disassociation attack. 802.11w thus fails to protect against all DoS attacks that are equivalent to the deauthentication and disassociation attacks.

Introducing protected deauthentication and disassociation frames in 802.11w leads to a deadlock vulnerability. If the PTKSA in the AP is deleted while the station still has a valid PTKSA, then the station is not able to recover. This is the result of the SA termination attack. The proposed solution to this vulnerability by TGw is the SA Query procedure. This procedure has a weakness: an attacker who is able to delete messages or perform radio frequency (RF) jamming attacks will still be able to create a deadlock by sending an association request, then deleting the SA queries or perform RF jamming until the SA Query procedure times out. Message deletion in 802.11 networks is possible in the following way: the attacker listens for messages, then switches on the transmitter while the message is in transit to create a collision. Immediately after the collision, the attacker sends a MAC layer acknowledgment (ACK) to the sender. The sender thus assumes that the message was received, and no retransmission occurs. RF jamming attacks are even easier to perform. Since the association response from the AP contains the timeout value, the attacker knows exactly how long the jamming attack must last to result in a deadlock. The 802.11w drafts suggest a timeout value of around one second. An attacker can thus spend one second of RF jamming to permanently disconnect the station.

8 Proposal for a Robust Solution

The root cause of the DoS vulnerabilities in 802.11, both the previously known ones and the new vulnerability presented in this paper, is that 802.11 with amendments does not adhere to the first principle from [14]. The proposed solution adheres to this principle: To provide authentication from the very beginning. The challenge is how to do it, given the existing 802.11 standard with amendments. The creators of WEP did one thing right, their shared key authentication was performed as early as possible. This authentication, as well as the 802.11 open system authentication, is carried out using management frames of subtype authentication. Such frames have an important property, they contain an "Authentication Algorithm Number" field with a length of two bytes. Currently, only the values "0" (open system) and "1" (shared key) are used. This means that it is possible to add identifiers for new authentication methods.

Figure 9 shows the proposed authentication and key management procedure. The new authentication frame specification is the following: Add a new authentication algorithm number, "2", for RSN authentication. Add an RSN information

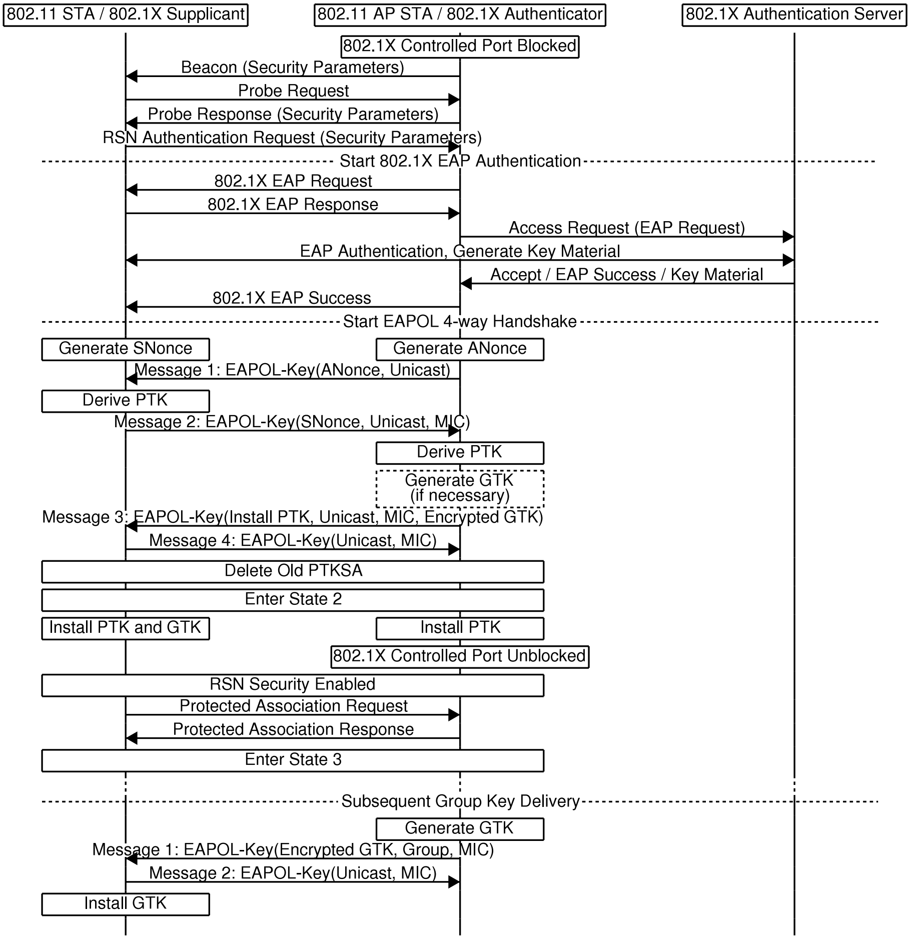

802.11 STA / 802.1X Supplicant
802.11 AP STA / 802.1X Authenticator
802.1X Authentication Server
802.1X Controlled Port Blocked
Beacon (Security Parameters)
Probe Request
Probe Response (Security Parameters)
RSN Authentication Request (Security Parameters)
Start 802.1X EAP Authentication
802.1X EAP Request
802.1X EAP Response
Access Request (EAP Request)
EAP Authentication, Generate Key Material
Accept / EAP Success / Key Material
802.1X EAP Success
Start EAPOL 4-way Handshake
Generate SNonce
Generate ANonce
Message 1: EAPOL-Key(ANonce, Unicast)
Derive PTK
Message 2: EAPOL-Key(SNonce, Unicast, MIC)
Derive PTK
Generate GTK (if necessary)
Message 3: EAPOL-Key(Install PTK, Unicast, MIC, Encrypted GTK)
Message 4: EAPOL-Key(Unicast, MIC)
Delete Old PTKSA
Enter State 2
Install PTK and GTK
Install PTK
802.1X Controlled Port Unblocked
RSN Security Enabled
Protected Association Request
Protected Association Response
Enter State 3
Subsequent Group Key Delivery
Generate GTK
Message 1: EAPOL-Key(Encrypted GTK, Group, MIC)
Message 2: EAPOL-Key(Unicast, MIC)
Install GTK

Fig. 9. The proposed solution for RSN authentication and key management. The authentication procedure is initiated when the station transmits an authentication frame with authentication algorithm number equal to 2 and a valid RSN information element. Management frames of subtype "authentication and key management" are used to encapsulate the 802.1X authentication messages, the 4-way handshake and the group key handshake. Note that the association request and response are protected.

element (security parameters) to the authentication frame. This enables the station to specify the authentication method and security parameters to be used in the authentication request.

The remaining issue is how to encapsulate the EAPOL messages used for authentication. This is solved by adding a new management frame subtype of Class 1, "authentication and key management". To remove all of the DoS vulnerabilities described in this paper, the 802.11i EAPOL authentication and key exchange messages are encapsulated in the authentication and key management frames, rather than in data frames. 802.11w should then be amended to also provide protection for authentication and key management, association request and association response frames. Note that for backwards compatibility, the use of data frames to transport EAPOL messages must still be supported as defined in 802.11i.

Finally, to avoid deadlocks, the PTKSA should not be terminated after a successful association, disassociation or deauthentication, but rather be replaced with a new PTKSA after a successful 4-way handshake. If the protected association procedure fails, both the station and AP should perform the deauthentication procedure.

The construction outlined above can be backwards compatible with 802.11 with amendments, as noted. However, as long as backwards compatibility is preserved, the network will still be vulnerable to the authentication attack described in subsection 5.4. A transitional workaround for this is that the AP maintains a list of stations that have been successfully authenticated using the new authentication method, and that authentication requests for open system or shared key authentication for these stations are ignored. If backwards compatibility is discarded, this is not an issue, since an attacker will not be able to successfully authenticate.

9 Conclusion

All of the attacks presented in this paper were carried out using off the shelf hardware and freely available software. No software or hardware modification was necessary, so any person with access to a laptop computer and an Internet connection should be able to replicate these experiments or carry out actual DoS attacks.

Although the SA termination vulnerability from forged association frames has been addressed in recent draft versions of 802.11w, implementations of early drafts are still vulnerable. Until these have been updated, a network with 802.11w enabled is *more vulnerable* to DoS than a network without. Since the only purpose of 802.11w at the moment is to protect against DoS, a sound recommendation would be to disable it until a solution for this vulnerability is provided.

The SA Query procedure proposed as a solution to the SA termination vulnerability does not protect against an attacker who is able to delete messages or perform RF jamming attacks. Due to the severity of this vulnerability, the author strongly recommends that a more robust solution, such as the one proposed in section 8, is adopted.

The 802.11w drafts do not, as far as the author is aware of, address the authentication attack of subsection 5.4. If protection against all DoS attacks with efficiency and feasability equivalent to the disassociation attack is a goal of TGw, the proposed solution from this paper should be included in the 802.11w amendment.

Acknowledgements

The author would like to thank Stig F. Mjølsnes for valuable help and advice, Jing Xie for suggesting how to design a robust solution, and the anonymous reviewers of this paper for their helpful comments.

References

1. Aircrack-ng, http://www.aircrack-ng.org
2. Wireshark, http://www.wireshark.org
3. Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., Levkowetz, H.: Extensible Authentication Protocol (EAP), IETF RFC 3748 (2004)
4. Bellardo, J., Savage, S.: 802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions. In: SSYM 2003: Proceedings of the 12th conference on USENIX Security Symposium (2003)
5. Bittau, A., Handley, M., Lackey, J.: The Final Nail in WEP's Coffin. In: SP 2006: Proceedings of the 2006 IEEE Symposium on Security and Privacy, pp. 386–400 (2006)
6. Epstein, J.: SA Teardown Protection for 802.11w, IEEE TGw DCN 2441, Rev 3 (2007)
7. Epstein, J.: SA Teardown Protection, IEEE TGw DCN 2461, Rev 8 (2007)
8. Fluhrer, S., Mantin, I., Shamir, A.: Weaknesses in the Key Scheduling Algorithm of RC4. In: Proceedings of the 4th Annual Workshop on Selected Areas of Cryptography, pp. 1–24 (2001)
9. The Institute of Electrical and Electronics Engineers, Inc.: IEEE Std 802.11-1999. IEEE, New York (1999)
10. The Institute of Electrical and Electronics Engineers, Inc.: IEEE Std 802.11i-2004. IEEE, New York (2004)
11. The Institute of Electrical and Electronics Engineers, Inc.: IEEE Std 802.11X-2004. IEEE, New York (2004)
12. The Institute of Electrical and Electronics Engineers, Inc.: IEEE P802.11w/D3.0. IEEE, New York (2007)
13. The Institute of Electrical and Electronics Engineers, Inc.: IEEE Std 802.11-2007. IEEE, New York (2007)
14. Meadows, C.: A Formal Framework and Evaluation Method for Network Denial of Service. In: IEEE Computer Security Foundations Workshop, p. 4 (1999)
15. Tews, E., Weinmann, R.P., Pyshkin, A.: Breaking 104 Bit WEP in Less Than 60 Seconds. In: Cryptology ePrint Archive, Report 2007/120 (2007)

Fast Packet Classification Using Condition Factorization

Alok Tongaonkar, R. Sekar, and Sreenaath Vasudevan

Stony Brook University

Abstract. Rule-based packet classification plays a central role in network intrusion detection systems such as Snort. To enhance performance, these rules are typically compiled into a *matching automaton* that can quickly identify the subset of rules that are applicable to a given network packet. The principal metrics in the design of such an automaton are its size and the time taken to match packets at runtime. Previous techniques for this problem either suffered from high space overheads (i.e., automata could be exponential in the number of rules), or matching time that increased quickly with the number of rules. In contrast, we present a new technique that constructs polynomial size automata. Moreover, we show that the matching time of our automata is insensitive to the number of rules. Our experimental results demonstrate substantial improvements in space requirements, as well the runtime of Snort.

1 Introduction

Given a network packet p and a set of *signatures* (which capture a set of conditions on the content of network packets), the problem of *packet classification* is that of identifying the subset of signatures that match p. It is the central computation performed in network intrusion detection systems (NIDS) such as Snort [14].

A naive technique for packet classification is that of sequentially matching each signature against an incoming packet. The performance of such a technique degrades linearly with the number of signatures. Since the number of signatures used in NIDS applications is typically large (e.g., Snort rule sets consist of several thousand rules), this naive technique will not scale to even moderate speed networks.

A natural way to speed up classification is to build a search-tree-like data structure that can be used to narrow down the set of signatures that are applicable to a packet, and then match the packet sequentially against each of the signatures in this subset. A common technique is to base the search tree on a small set of packet attributes that are present in almost all rules, e.g., Snort (versions 2.x) uses a search tree that first branches on the protocol (e.g., IP or ICMP), and then on source and destination ports (for TCP- and UDP-related rules).

By limiting to a small number of predefined attributes, we can simplify the search-tree construction algorithm, and also ensure that the tree is small in size. But the drawback is that the number of signatures that remain applicable at a

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 417–436, 2009.

leaf node can be substantial. As a result, the sequential matching phase can still take significant time. To further reduce this time, it would be desirable to build search trees that can make use of all (or most) of the attributes that occur in signatures, instead of limiting to a small number of predefined attributes. However, building such search trees becomes complex because some of the attributes may not be present in all signatures. Consider a search tree node that examines such an attribute. If node has two children, signatures that do not examine this attribute would need to be duplicated across these children. Repeated duplication leads to search trees whose size, in the worst-case, is *exponential in the number of signatures* [16].

In contrast with previous techniques, we develop a new, systematic approach that ensures a polynomial bound on the size of the search tree. In addition to space-reductions, our approach improves classification speed using a novel technique called *condition factorization* that breaks down tests involving packet fields in such a manner as to expose commonalities across different types of tests such as equality tests, inequality tests, tests involving bit-masking operations, etc. Our experimental results indicate an overall performance gain of 30% for Snort. Moreover, as compared to previous techniques for constructing packet classification search trees such as Snort-NG [9], our techniques lead to search trees that are tens to hundreds of times smaller. Below, we present an overview of our approach and summarize its key contributions.

1.1 Overview of Approach and Contributions

- In Section 2, we formalize the problem of packet classification as applicable to intrusion detection systems.
- In Section 3, we develop the concept of *condition factorization* that provides the foundation for the optimizations developed in this paper. Condition factorization is based on the notion of a *residual* of a condition with respect to another. Intuitively, if we think of logical conjunction as analogous to the product operation on integers, then residuals are analogous to the division operation. Just as division provides the basis for finding common factors among integers, residuals provide the basis for "factorizing" complex conditions originating from different rules so as to "share" the testing of their common parts.
- In Section 4 we present our automaton[1] construction algorithm. Condition factorization is the core operation behind this algorithm, and it contributes directly to two key optimizations:
  - It can reason about the relationships between the typical operations that arise in rules (e.g., equalities, inequalities, disequalities, and bit-masking operations) and leverage them to avoid *semantically redundant tests* even if they aren't syntactically identical. It is more general than the techniques developed in BPF+[3] for eliminating semantic redundancies — our technique proactively creates opportunities for sharing computation, whereas BPF+ is limited to checking whether previously performed tests obviate the need for a new test.

[1] Henceforth, we use the term "automaton" instead of the term "search-tree."

  - By working with residuals of rules, our automaton construction algorithm can recognize equivalence between automata states even before constructing the descendant states. Such *direct construction* is important, since a tree automaton is usually much larger (in theory, exponentially larger) than a DAG automaton. As a result, techniques that minimize tree automaton into a DAG automaton are bound to significantly increase space and time needed for automata construction.
- In Section 5, we present several additional techniques for building space- and time-efficient automata:
  - In Section 5.1, we develop the notion of a *discriminating test.* If such tests are selected at every state of the automaton, its size would be polynomial in the size of input rules. Unfortunately, discriminating tests may not always exist, which can lead to an explosion in automaton size. We therefore present a new technique in Section 5.2 that guarantees polynomial space bounds (where the degree of the polynomial can be user-specified) by trading off some determinism. We point out that this theoretical possibility of nondeterminism wasn't observed in our experiments. Thus, our technique was able to guarantee quadratic worst-case space requirement, without incurring, in practice, the performance penalties associated with nondeterminism.
  - In Section 5.3, we develop the notion of *benign nondeterminism*, which enables the introduction of nondeterministic branches in the automaton *without any increase in matching times.* Our experiments indicate dramatic reductions in automata size as a result of this technique.
- In Section 6, we describe our implementation, followed by an experimental evaluation in Section 7. Our technique achieves over a *10-fold reduction* in space requirements as compared to previous packet classification techniques for NIDS, while improving matching times. Moreover, the experimentally observed matching time remains virtually constant, regardless of the number of rules. In contrast, previous techniques experience a significant slowdown as the number of rules are increased. Our experiments also show that each of the techniques presented in previous sections contributes to significant reduction in space requirements.
- Related work is described in Section 8, followed by concluding remarks in Section 9.

We point out that string-matching and regular-expression matching techniques are orthogonal to the techniques developed in this paper. In particular, a common strategy used in NIDS is to build a search-tree based on packet fields. At each leaf of this search-tree, a string-matching (or finite-state) automaton corresponding to the signatures S associated with this leaf is built. In the case of Snort, an Aho-Corasick automaton [1] is used, which identifies a subset of S' of S whose longest string matches the current network packet. Our techniques reduce the size of S by building a search-tree based on most packet fields, and hence the size of S' is also correspondingly reduced, which translates into faster times for the final (sequential) matching phase.

2 Preliminaries

In the rest of this paper, we use the term *filter* to refer to signatures. We associate a label to identify a filter.

Definition 1 (Tests, Filters and Priorities). *A* ***test*** *involves a variable x and one or two constants (denoted by c) and has one of the following forms.*

- Equality tests of the form $x = c$
- Equality tests with bitmasks of the form $x \& c_1 = c$
- Disequality tests of the form $x \neq c$
- Disequality tests with bitmasks of the form $x \& c_1 \neq c$
- Inequality tests of the form $x \leq c$ or $x \geq c$

A ***filter*** *F is a conjunction of tests.*

An example of a filter, as defined above, is

$$(dport = 22) \wedge (sport \leq 1024) \wedge (flags \& \texttt{0xb} = \texttt{0x3})$$

We exclude more complex conditions that don't satisfy the definition of a filter, e.g.,

$$(sport + dport < 1024) \wedge (sport < ttl),$$

since they do not seem to arise in practice.

A filter F can be "applied" to a network packet p, denoted $F(p)$, by substituting variables, which denote the names of packet fields, with the corresponding values from p. We define the notion of matching based on whether the filter evaluates to *true* after this substitution.

Definition 2 (Matching). *For a set $\mathcal{F}$ of filters, we say that $F \in \mathcal{F}$* ***matches a packet*** *p if $F(p)$ is true. The* ***match set*** *of p, denoted $\mathcal{M}_{\mathcal{F}}(p)$ consists of all filters that match p.*

To illustrate matching, consider the following filter set $\mathcal{F}$:

- $F_1 : (icmp\_type = ECHO)$
- $F_2 : (icmp\_type = ECHO\_REPLY) \wedge (ttl = 1)$
- $F_3 : (ttl = 1)$

Also consider an *icmp echo* packet p_1 and an *icmp echo reply* packet p_2, both having a *ttl* of 1. For these filters and packets, F_1 matches p_1, F_2 matches p_2, and F_3 matches both. As a result, $\mathcal{M}_{\mathcal{F}}(p_1) = \{F_1, F_3\}$ and $\mathcal{M}_{\mathcal{F}}(p_2) = \{F_2, F_3\}$

Examples of *packet-matching automata* (also known as matching or classification automata) for the above filter set are shown in Figures 1 and 2. Figure 1 shows a *deterministic automaton*, in which all of the transitions from any automaton state are mutually exclusive. A *non-deterministic automaton* is shown in Figure 2, where the transitions may not be mutually exclusive. We make the following observations about the structure of matching automata:

- All but one of the transitions from each state are labeled with a *test* as defined above; the remaining (optional) transition, called an "other" transition, is labeled with a more complex condition C as follows:

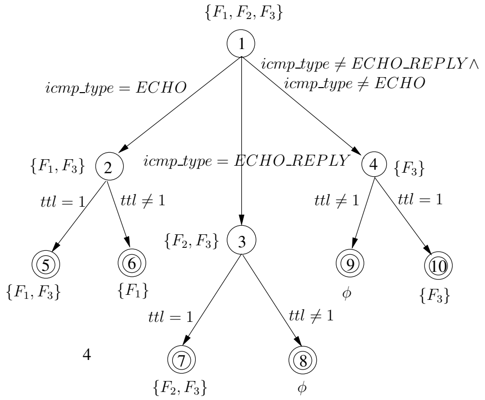


Fig. 1. A deterministic matching automaton

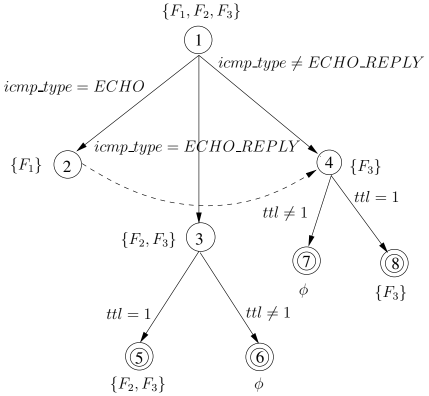


Fig. 2. A non-deterministic matching automaton

  - In a non-deterministic automaton, C is the conjunction of negations of *a subset* of the tests on the rest of the transitions, e.g., the third transition from the start state in Figure 2.
  - In a deterministic automaton, C is the conjunction of negations of *all* the tests on the rest of the transitions, e.g., the third transition from the start state in Figure 1. In this case, the "other" transition is mutually exclusive with the rest of the transitions, and hence is also called an "else" transition.

- The transitions from each automaton state are *simultaneously distinguishable,* i.e.,
  - apart from the "*other*"-transition, the tests on the rest of the transitions are mutually exclusive
  - it is possible to determine, using a single operation with $O(1)$ expected time complexity, which of the transitions out of a state is applicable to a given packet.
- Each final state S correctly identifies the match set corresponding to any packet satisfying all the tests along a path from the start state to S.

Note that non-determinism has a runtime cost, as it needs to be simulated using backtracking. For instance, consider a packet that satisfies the $icmp\_type = ECHO$ condition on the first transition from the start state of Figure 2. This packet is also compatible with the condition $icmp\_type \neq ECHO\_REPLY$ on the third transition from the start state. Thus, after exploring down the first transition, it is necessary to explore down the third transition as well. This need for backtracking is depicted in Figure 2 using a dotted transition.

3 Condition Factorization

In this section, we introduce the novel concept of condition factorization. It refers to the process of decomposing filters into combination of more primitive tests — a process that is intuitively similar to factorization of integers. This decomposition exposes those primitive tests that are common across different tests, and thus enables shared computation of these common primitive tests.

The basis for condition factorization is the residue operation defined below. It is analogous to integer division. Suppose that we want to determine if there is a match for a filter C_1. Also assume that we have so far tested a condition C_2. A residue captures the additional tests that need to be performed at this point to verify C_1.

Definition 3 (Residue). *For conditions C_1 and C_2, the* ***residue*** *C_1/C_2 is another condition C_3 such that:*

(1) $C_2 \wedge C_3 \Rightarrow C_1$, and
(2) $C_1 \wedge C_2 \Rightarrow C_3$.

For a filter set, $\mathcal{F}/C = \{F/C | F \in \mathcal{F} \wedge F/C \neq false\}$.

Ideally, C_3 would be the weakest condition such that (1) holds. In practice, however, we may not want the minimal condition since it may be expensive to compute, or be inefficient to use, e.g., may contain many disjunctions. For this reason, we do not require C_3 be the weakest such condition. But C_3 shouldn't be too strong, or else we may miss matches for C_1. This motivates the condition (2) above.

The rules in Figure 3 specify how to compute residues on tests. In the figure, the notation $\overline{x}$ denotes bit-wise complement of x, while & denotes bit-wise "and" operation. In addition, inequalities are expressed using interval constraints, e.g., $x \leq 7$ is represented as $x \in [-\infty, 7]$, if x is an integer-valued variable. Note that a single interval constraint can represent a pair of inequalities involving a single variable, e.g., $(x \leq 7) \wedge (x > 3)$ can be represented as $x \in [4, 7]$.

For any pair of tests T_1 and T_2, the first row in the table that matches the structure of T_1 and T_2 yields the value of T_1/T_2. We illustrate residue computation using several examples:

- $(x \neq a)/(x = a)$ is $false$, as given by the second row in the table (which defines $T/\neg T$).
- $(x = 5)/(x\&\mathtt{0x3} \neq 1)$ is $false$, as given by the 5th row.

T_1	T_2	T_1/T_2	Conditions
T	T	*true*	
T	$\neg T$	*false*	
T	$x = c$	$T[x \leftarrow c]$	
$x = c$	$x \,\&\, c_1 = c_2$	$x \,\&\, \overline{c_1} = c \,\&\, \overline{c_1}$ *false*	$c \,\&\, c_1 = c_2$ $c \,\&\, c_1 \neq c_2$
$x = c$	$x \,\&\, c_1 \neq c_2$	*false*	$c \,\&\, c_1 = c_2$
$x = c$	$x \in [c_1, c_2]$	*false*	$c \notin [c_1, c_2]$
$x \neq c$	$x \,\&\, c_1 = c_2$	$x \,\&\, \overline{c_1} \neq c \,\&\, \overline{c_1}$ *true*	$c \,\&\, c_1 = c_2$ $c \,\&\, c_1 \neq c_2$
$x \neq c$	$x \,\&\, c_1 \neq c_2$	*true*	$c \,\&\, c_1 = c_2$
$x \neq c$	$x \in [c_1, c_2]$	*true*	$(c < c_1) \vee (c > c_2)$
$x \in [c_1, c_2]$	$x \in [c_3, c_4]$	*true* $x \in [-\infty, c_2]$ $x \in [c_1, \infty]$ $x \in [c_1, c_2]$ *false*	$c_1 \leq c_3 \leq c_4 \leq c_2$ $c_1 \leq c_3 \leq c_2 \leq c_4$ $c_3 \leq c_1 \leq c_4 \leq c_2$ $c_3 \leq c_1 \leq c_2 \leq c_4$ $(c_2 < c_3) \vee (c_4 < c_1)$
$x \in [c_1, c_2]$	$x \,\&\, c_3 = c_4$	*false*	$c_4 > c_2$
$x \,\&\, c_1 = c_2$	$x \,\&\, c_3 = c_4$	$x \,\&\, (c_1 \,\&\, \overline{c_3}) = (c_2 \,\&\, \overline{c_3})$ *false*	$c_2 \,\&\, c_3 = c_1 \,\&\, c_4$ otherwise
$x \,\&\, c_1 = c_2$	$x \in [c_3, c_4]$	*false*	$c_2 > c_4$
$x \,\&\, c_1 \neq c_2$	$x \,\&\, c_3 = c_4$	$x \,\&\, (c_1 \,\&\, \overline{c_3}) \neq (c_2 \,\&\, \overline{c_3})$ *true*	$c_2 \,\&\, c_3 = c_1 \,\&\, c_4$ otherwise
$x \,\&\, c_1 \neq c_2$	$x \in [c_3, c_4]$	*true*	$c_2 > c_4$
T	T'	T	

Fig. 3. Computation of Residue on Tests

- for $(x = 5)/(x\&\texttt{0x3} \neq 0)$, 5th row is no longer applicable since the condition $c\&c_1 = c_2$ does not hold. (Here, $c = 5$, $c_1 = \texttt{0x3}$, and $c_2 = 0$.) Hence the applicable row is the last row, which yields $(x = 5)/(x\&\texttt{0x3} \neq 0) = (x = 5)$. The result is understandable: although the two conditions are compatible with each other, the test $x\&\texttt{0x3} \neq 0$ does not contribute to proving $x = 5$.
- $(x \in [1, 10])/(x \neq 5)$ is also given by the last row to be $(x \in [1, 10])$.

Note that the *minimal* residue in the last example would be $(x \in [1, 4]) \vee (x \in [6, 10])$. In this sense, Figure 3 makes approximations in computing residues. Intuitively, we make this approximation since there does not seem to be any way to evaluate $(x \in [1, 4]) \vee (x \in [6, 10])$ more efficiently than $(x \in [1, 10])$.

In general, approximations such as those used above have the potential to lead our matching algorithm to perform multiple tests that have some semantic overlap. However, the first line in Figure 3 ensures that two syntactically identical tests would never be performed.

To illustrate residues on filter sets, consider

$$\mathcal{F} = \{F_1 : (x = 5),\ F_2 : (x = 7),\ F_3 : (x < 10)\}.$$

Then

- $\mathcal{F}/(x = 5) = \{F_1 : \mathit{true},\ F_3 : \mathit{true}\}$
- $\mathcal{F}/(x < 7) = \{F_1 : (x = 5),\ F_3 : \mathit{true}\}$

Finally, we specify how to compute residues on more complex conditions that are formed using conjunction and disjunction operations on tests:

- $(C_1 \oplus C_2)/C_3 = (C_1/C_3) \oplus (C_2/C_3),$ for $\oplus \in \{\wedge, \vee\}$
- $C_1/(C_2 \wedge C_3) = (C_1/C_2)/C_3$

We have ignored the case where the second operand to the residue operator contains a disjunction, since this case does not arise in our automata construction algorithm. Using this definition, we can see that:

- $((x > 2) \vee (y > 7))/(x = 5)$ is *true*, and
- $((x > 2) \wedge (y > 7))/(x = 5)$ is $(y > 7)$.

4 Matching Automata Construction

Our algorithm *Build* for constructing a matching automata is shown in Figure 4. *Build* is a recursive procedure that takes an automaton state s as its first parameter, and builds the subautomaton that is rooted at s. It takes two other parameters: (i) the *match set* $\mathcal{M}_s$ that consists of all filters for which a match can be announced at s, and (ii) the *candidate set* $\mathcal{C}_s$ that consists of filters that haven't completed a match, but future matches can't be ruled out either, i.e., matches for these filters will be reported at some of the descendants of s. To illustrate the concepts of match and candidate sets, we have annotated the final states in Figures 1 and 2 with match sets, and non-final states with the union of match and candidate sets.

We maintain only the residuals of the original filters in $\mathcal{C}_s$ and $\mathcal{M}_s$, after factoring out the tests performed on the path from the root of the automaton to the state s. For example, in Figure 1, at state 2, we have completed a match for F_1, and hence its match set is $\{F_1 : \mathit{true}\}$. Note that the condition component of F_1 has become *true* since we computed the residue of the original condition (i.e., $(\mathit{icmp\_type} = \mathit{ECHO})$) with respect to the condition $(\mathit{icmp\_type} = \mathit{ECHO})$ on the path from the automaton root to state 2. In addition, note that we can rule out a match for F_2 at this state, but a match for F_3 is still possible. Thus, the candidate set for this state is $\{F_3 : (\mathit{ttl} = 1)\}$.[2]

[2] $\mathcal{M}_s$ and $\mathcal{C}_s$ can be formally defined as follows. Let $\mathcal{P}_s$ denote the conjunction of tests on the path from the start state of the automaton to the state s. Then $\mathcal{M}_s = \{F \in \mathcal{F}/\mathcal{P}_s | (F = \mathit{true})\}$. Similarly, $\mathcal{C}_s = \{F \in \mathcal{F}/\mathcal{P}_s | (F \neq \mathit{true})\}$

```
1.  procedure $Build(s, \mathcal{M}_s, \mathcal{C}_s)$
2.    if $\mathcal{C}_s$ is empty /* No more filters to match */
3.        then $match[s] = \mathcal{M}_s$ /* Annotate final state with match set */
4.    else
5.        $(D, \mathcal{T}) = select(\mathcal{C}_s)$ /* $T_i \in \mathcal{T}$ is tested on $i$th transition */
              /* $d_i \in D$ indicates if this transition is deterministic */
6.        $T_o = \{\bigwedge_{d_i \in D | d_i = true} \neg T_i\}$
              /* Compute test corresponding to the "other"-transition */
7.        for each $T_i \in (\mathcal{T} \cup \{T_o\})$ do
8.            $\mathcal{C}_i = \mathcal{C}_s / T_i$
9.            if $((T_i \neq T_o) \wedge \neg d_i)$ then $\mathcal{C}_i = \mathcal{C}_i - \mathcal{C}/T_o$ endif
                  /* For a non-deterministic transition, do not duplicate */
                  /* filters from the "other" branch */
10.           compute $\mathcal{M}_{s_i}$ and $\mathcal{C}_{s_i}$ from $\mathcal{C}_i$ and $\mathcal{M}_s$
11.           if a state $s_i$ corresponding to $(\mathcal{C}_{s_i}, \mathcal{M}_{s_i})$ isn't present
12.               create a new state $s_i$
13.               $Build(s_i, \mathcal{M}_{s_i}, \mathcal{C}_{s_i})$
14.           endif
15.           create a transition from $s$ to $s_i$ on $T_i$
16.       end
17.   endif
```

Fig. 4. Algorithm for Constructing Matching Automaton

A final state is characterized by the fact that there are no more filters left in $\mathcal{C}_s$. This condition is tested at line 2, and s is marked final, and is annotated to indicate $\mathcal{M}_s$ as its match set. If the condition at line 2 isn't satisfied, then the construction of automaton is continued in lines 5–16. First, a procedure *select* (to be defined later) is used at line 5 to identify a set of tests $T_1, ..., T_k$ that would be performed on the transitions from s. This procedure also indicates whether T_i is going to be a deterministic transition or not: in the former case d_i is set to $true$, while in the latter case, $d_i = false$. Based on which T_i are deterministic, the condition T_o associated with the "other"-transition is computed on line 6: $\neg T_i$ is included in T_o iff T_i is to be a deterministic transition.

The actual transitions are created in the loop at line 7–16. At line 8, we compute the subset $\mathcal{C}_i$ of filters in $\mathcal{C}_s$ that are compatible with T_i. However, if this is going to be a nondeterministic transition, then a match would be tried down the transition labeled T_i and then subsequently down the "other"-transition. For this reason, we can eliminate from $\mathcal{C}_i$ any filter that will be considered on the "other"-transition. This elimination is performed on line 9. At line 10, $\mathcal{M}_{s_i}$ and $\mathcal{C}_{s_i}$ for the new state s_i are computed.

Since the behavior of *Build* is determined entirely by the parameters $\mathcal{C}_s$ and $\mathcal{M}_s$, two invocations of *Build* with the same values of these parameters will yield identical subautomata. Hence a check is made at line 11 to examine if an automaton state already exists corresponding to $\mathcal{C}_{s_i}$ and $\mathcal{M}_{s_i}$, and if not, a new state is created at line 12, and *Build* recursively invoked on this state. Finally, a transition to this state is created at line 15.

5 Improving Automata Size

The algorithm presented in the last section incorporated two main optimizations to reduce automaton size and matching time, both derived from our definition of condition factorization: detecting and sharing equivalent states, and avoiding repetition of (semantically) redundant tests. In this section, we present techniques for realizing the *select* function that yields significant additional reduction in automata size.

Although our experimental evaluation considers the number of automaton states as a measure of its size, for simplifying mathematical analysis, our discussion in this section will use the automaton breadth as the size metric. Since the automaton is acyclic, and since tests are never repeated, it can be shown that the total number of automaton states can, in the worst case, be at most S times its breadth, where S is the number of distinct tests across all the filters[3].

5.1 Discriminating Tests

Definition of *select* amounts to determining the test that should be performed at a particular state of the automaton. Since the test identifies the packet field to be examined, *select* can be viewed as defining an order of examination of packet fields. Not all orders of examination may be acceptable, since some packet fields (e.g., the protocol field) may need to be examined before others (e.g., the port field). We use a type system similar to packet types [4] that captures such ordering constraints among tests. Our implementation of *select* ensures that these constraints are respected.

The simplest approach for defining *select* is to test the fields in the order of their occurrence in a network packet, as done in some of the previous works [2,5]. We call such a traversal order as *left-to-right traversal* and refer to an automaton using this traversal order as *L-R automaton.* A better strategy, called *adaptive traversal,* was first proposed in the context of term-matching [16], and was then generalized to deal with binary data in [7]. In the terminology of this paper, an adaptive traversal would select a set of tests $\mathcal{T}$ at an automaton state s as follows. It identifies a packet field x that occurs in every filter in $\mathcal{C}_s$. (If no such field can be found, it falls back to another choice, e.g., choosing the left-most field that hasn't yet been examined.) Now, $\mathcal{T}$ consists of all tests on x that occur in any of the filters in $\mathcal{C}_s$.

Since adaptive traversal was developed in a context where the tests were all restricted to be simple equalities with constants, it is easy to see that the set $\mathcal{T}$ described above consists of tests that can be simultaneously distinguished[4], and hence can form the transitions from s. Moreover, it has been shown [16] that, as compared to other choices, this choice of transitions will simultaneously reduce the automaton size as well as matching time. Unfortunately, none of these hold in

[3] In practice, the factor is closer to average size of filters, which can be significantly smaller than S.

[4] Recall that simultaneous distinguishability refers to the ability to identify the matching transition in $O(1)$ expected time.

the more general setting of packet matching, where disequalities and inequalities also need to be handled. For instance, consider a candidate set that consists of two filters ($x \neq 25$) and ($x < 1024$). These tests are not simultaneously distinguishable. Moreover, neither of these tests contributes towards verifying a match with the other. More generally, it can be shown that, in the presence of disequality and inequality tests, the choices that decrease automaton size do not necessarily decrease matching time (and vice-versa). We therefore focus first on a criterion for reducing automaton size.

Definition 4 (Discriminating Set). *A set $\mathcal{T}$ of conditions is said to be a **discriminating set** for a filter set $\mathcal{F}$ iff for every $F \in \mathcal{F}$ there exists at most one $T \in \mathcal{T}$ such that F belongs to the candidate set of $\mathcal{F}/T$.*

The set $\mathcal{T} = \{x = 5, x = 6, (x \neq 5) \wedge (x \neq 6)\}$ is discriminating for the filter set $\mathcal{C} = \{x = 5, x = 6, x > 7\}$, but not for $\{x = 6, x > 4\}$. This means if we create 3 outgoing transitions corresponding to the three tests in $\mathcal{T}$ from an automata state s with the candidate set $\mathcal{C}$, none of the filters in $\mathcal{C}$ will be duplicated among the children of s. As a result, in an automaton that uses only discriminating tests, the candidate sets (as well as the match sets) associated with the leaves will be disjoint. Since there are at most n disjoint subsets of a set of size n, it immediately follows that any automataon that is based entirely on discriminating tests will have at most $O(n)$ breadth.

5.2 Ensuring Polynomial-Size Automata

Since discriminating tests may not always exist, it may be necessary to choose non-discriminating tests. This choice introduces overlaps among the candidate sets of sibling states in the automaton. These overlaps, in turn, mean that at any level in the automaton, there may be as many as 2^n distinct candidate sets. Thus, the breadth of the automaton can become exponential in the number of filters. Exponential *lower bounds* have previously been established even in the simple case where all tests are restricted to be equalities [16]. Although some of the previously developed techniques can avoid such explosion, this has been accomplished at the cost of introducing significant backtracking at runtime [11,5,2,3], especially for the kinds of filters that occur in the context of intrusion detection. Other techniques avoid exponential size by introducing $O(n)$ operations for each transition at runtime, as they require runtime maintenance of match sets [13,7]. With large filter sets that are often found in enterprise NIDS, $O(n)$ time complexity for transitions becomes unacceptable.

We present a new technique that can provide a polynomial size bound, while limiting nondeterminism in practice. Indeed, any desired polynomial bound $P(n)$ can be achieved by our technique. However, by using a larger bound, e.g., n^2 instead of $n \log n$, one can obtain deterministic automata in almost all cases.

Our technique is based on the observation that the breadth of subautomaton rooted at s can be captured, in terms of the sizes of candidates sets associated with s and its children, using the recurrence

$$B(|\mathcal{C}_s|) = \sum_{i=1}^{k} B(|\mathcal{C}_{s_i}|),$$

where $B(1) = 1$. Let $P(n)$ be the desired polynomial on n that bounds the automaton size. Based on the above recurrence, we can show, by induction on the height of s that the bound will be satisfied as long as the following condition holds at every state s of the automaton.

$$P(|\mathcal{C}_s|) \geq \sum_{i=1}^{k} P(|\mathcal{C}_{s_i}|) \tag{1}$$

By selecting tests that satisfy this constraint, our implementation of *select* ensures that the automaton size will be $O(P(n))$. If no such test can be found, our technique picks a test that comes the closest to satisfying this constraint, and then makes some of the outgoing transitions nondeterministic so as to ensure that sizes of candidate sets associated with the descendant automaton states satisfy the above constraint. Recall from line 9 of *Build* that making a test $\mathcal{T}_i$ nondeterministic enables us to avoid overlaps between $\mathcal{C}_i$ and $\mathcal{C}_o$. So, our algorithm makes one or more transitions out of an automaton state nondeterministic until Inequality 1 is satisfied. In our implementation, we have set $P(n)$ to be n^2, which guarantees a quadratic worst-case automaton size.

To understand the importance of the above technique, note that a purely deterministic technique ensures good performance at runtime, but risks catastrophic failure on large rulesets that cause an exponential blow up — memory will be exhausted in that case and hence the ruleset can't be supported. In contrast, our approach converts this catastrophic risk into the less serious risk of performance degradation. Unlike previous techniques for space reduction that led to increases in runtime in practice, performance degradation remains a theoretical possibility with our technique, rather than something observed in our experiments. (This is because of the fact that with the rulesets we have studied in our experiments, the quadratic bound was never exceeded, and hence nondeterminism was not introduced.)

5.3 Benign Nondeterminism

For our final space-reduction technique, we define the concept of benign nondeterminism, which enables us to benefit from the space-savings enabled by non-determinism *without incurring any performance penalties.* It is based on the following notion of *independence* among filter sets.

Definition 5 (Independent Filters). *Two filters F_1 and F_2 are said to be* ***independent*** *of each other if $F_2/T = F_2, \forall T \in F_1$, and $F_1/T = F_1, \forall T \in F_2$. $\mathcal{F}_1$ and $\mathcal{F}_2$ are said to be independent if $\forall F_1 \in \mathcal{F}_1, \forall F_2 \in \mathcal{F}_2$, F_1 and F_2 are independent.*

Suppose that there is a filter set $\mathcal{F}$ that can be partitioned into two independent subsets $\mathcal{F}_1$ and $\mathcal{F}_2$. We can then build separate automata for $\mathcal{F}_1$ and $\mathcal{F}_2$. Packets can now be matched using the first automaton and then the second one. From the above definition, it is clear that the tests appearing in the two automata are completely disjoint, and hence no decrease in runtime can be achieved by constructing a single automaton for $\mathcal{F}$.

Our experiments show that the above technique leads to dramatic reductions in space usage. The intuition for this is as follows. If F_1 and F_2 are independent, then a packet may match F_1, F_2, both, or neither. A deterministic automaton must have a distinct leaf corresponding to each of these possibilities. Extending this reasoning to independent filter sets, if an automaton for the set $\mathcal{F}_1$ has k_1 states, and the automaton for $\mathcal{F}_2$ has k_2 states, then a deterministic automaton for $\mathcal{F}_1 \cup \mathcal{F}_2$ will have $k_1 * k_2$ states. In contrast, using benign non-determinism, the size is limited to $k_1 + k_2$. If there are m independent filter sets, then the use of benign nondeterminism can reduce the automaton size from a product of m numbers to their sum.

The second reason for significant reductions in practice, is as follows. After examining some of the fields that are common across many rules, as we get closer to the automaton leaf, independent sets arise frequently. For instance, we may be left with one set that examines only the destination port, another set that examines only the source port, yet another set that examines only the destination network, and so on. Thus, independent rule sets tend to arise frequently, and lead to massive increases in space usage if they are not recognized and exploited using our benign non-determinism technique.

There is a simple algorithm for checking if $\mathcal{F}$ contains two independent subsets. First, partition $\mathcal{F}$ into singleton subsets corresponding to each rule. Now, these subsets are taken two at a time, and merged if they are *not* independent. This process is repeated until no more merges are possible. If there are multiple subsets left at this point, then these subsets are independent.

To deal with benign non-determinism, the interface between *select* and *Build* needs to be extended so that the former can return a set of independent filter sets $\{\mathcal{F}_1, \ldots, \mathcal{F}_k\}$, instead of a test set. At this point, *Build* will create a k-way non-deterministic branch. On the ith branch, it will invoke $\mathit{Build}(s_i, \mathcal{F}_i, \mathcal{F}_i \cap \mathcal{M}_s)$.

6 Implementation: Putting It All Together

Our implementation first compiles the given filter set into an automaton using the *Build* algorithm. Residues were computed as specified in Table 3. Our *select* implementation proceeds as follows:

- *select* first attempts to find a discriminating test set (Section 5.1).
- if no discriminating test sets exist, it examines opportunities for benign non-determinism (Section 5.3).
- if neither of the above steps succeed, it returns a set of tests that achieves the polynomial size target specified, as described in Section 5.2.

In order to speed up *select*, our implementation starts by examining fields that occur in all filters in a candidate set, giving preference to those fields that contain primarily equality tests. Such fields have a high likelihood of yielding discriminating tests at which point *select* returns this set. As mentioned earlier, any constraints regarding the order of examination of fields are enforced by *select*.

Once the automaton is constructed, our compiler generates C-code corresponding to the automaton, which is then compiled into native code using a C-compiler. The code generation is straight-forward and not described in detail here, except to note that the code explicitly uses an if-then-else, a binary search, or a hash-based branching to implement transitions.

A runtime system is responsible for reading network packets and calling the generated code to perform matching. For experiments on network intrusion detection, our runtime system was essentially Snort, with modifications that were needed to integrate with our automata code.

7 Evaluation

The goal of our experimental evaluation is demonstrate performance improvements that can be gained in typical network intrusion detection systems as a result of using the packet classification techniques presented in the previous sections. To this end, we undertook two main experiments, both performed on a Linux system with 1.70Ghz Pentium 4 processor and 520MB memory, running CentOS-4.2 (Linux kernel 2.6.9).

7.1 End-to-End Performance Improvement of NIDS

In the first experiment, we replaced the simple packet classification used in Snort 2.6, the popular open-source NIDS, with our technique. Snort divides signatures into groups based on protocol, source port and destination port. For each such group, it extracts the longest string contained within the content-matching part of the signature, and builds an Aho-Corasick automaton for these signatures. At runtime, a simple packet classification technique is used to identify the rule group against which a packet needs to be matched. Then the content of the packet is matched using the Aho-Corasick automaton associated with this group. Since this automaton only considers the longest string from each signature, some

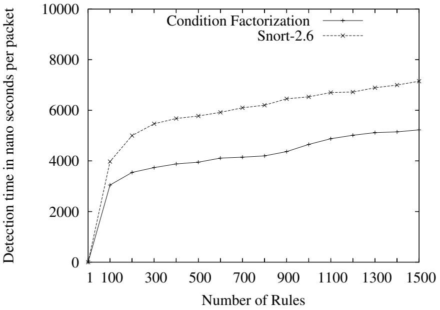


Fig. 5. Total Matching Time

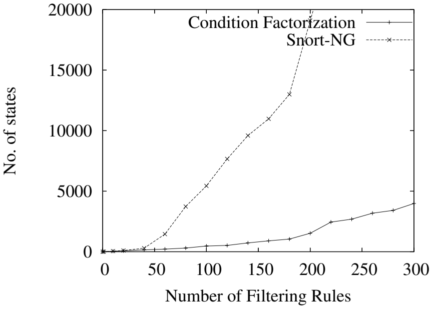


Fig. 6. Automaton Size for Snort Rules

of the signatures returned by this automaton may not really match the packet. (However, the automaton will always return a superset, not a subset of matching signatures.) Moreover, the signatures may contain complex conditions, e.g., a constraint on the distance between two strings within a signature. To handle these aspects, Snort performs a one-on-one match between a packet and each of the signatures returned by the automaton.

In our experiment, we replaced the first stage with the matching automaton constructed by our technique. At each leaf of this automaton, we replicate the technique used by Snort, i.e., we build an Aho-Corasick automaton to recognize the longest string contained in each of the signatures in the candidate set of the leaf[5]. Finally, a one-on-one match is perfomed between the signatures returned by this automaton and the network packet. Our implementation reuses almost all of Snort code, including the code for Aho-Corasick automaton, and the final one-on-one match. It only replaced the initial packet classification component. As a result, the performance improvements obtained by our technique are entirely due to the use of our sophisticated packet classifier.

We measured the total time taken by original Snort, and the version of Snort we modified to use our matching automaton. These times were computed for a 21-million packet trace collected at a University laboratory consisting of about 30 hosts. Since the firewall is fully open to the Internet (i.e., the traffic is not pre-screened by another layer of firewalls in the University or elsewhere), the traffic is a reasonable representative of what one might expect a NIDS to be exposed to. We used the default signature set that is shipped with Snort.

In this experiment, we observed that the one-on-one matching phase was invoked about 120M times in the original Snort, whereas it was invoked only 40M times with our packet classifier in place. This reduction in the number of one-on-one matches translates to about 30% reduction in the overall time taken by Snort.

Figure 5 shows the overall time taken by Snort with and without our modification, as we vary the number of rules. While the performance is nearly identical for small rule sets, it quickly increases to (and stabilizes at) about 30% at a few hundred signatures.

7.2 Improvement in Space Usage

In this experiment, we evaluated gains in space usage obtained using our packet classification technique. We first compared our technique against that of Snort-NG [9], which was the only other implementation of a sophisticated packet classifier that we are aware of that is applicable to NIDS like Snort. Snort-NG uses a different strategy from ours for eliminating redundant tests: they convert all tests into a canonical form so that semantically identical tests would also be

[5] In the presence of non-determinism, we needed to modify the above technique so as to avoid repetition of string-matching tests after backtracking. Specifically, we built the Aho-Corasick at the first non-deterministic node encountered on a root-to-leaf path in the automaton, and performed an intersection of the set of signatures returned by the Aho-Corasick with the signature sets of each of the matching leaves.

syntactically identical. However, tests in canonical form can in general be more expensive than the original test, e.g., in order to support tests on IP addresses that may sometimes involve bit-masking operations and at other times involve equality, they convert both tests into smaller tests that examine one bit of address at a time. Secondly, Snort-NG uses an entropy-based algorithm (instead of the criteria developed in Section 5 of this paper) to decide which packet field to test at each node. These factors lead to significant differences in the sizes of the automaton constructed, as illustrated in Figure 6.

We focused this evaluation on packet classification, and ignored the content-matching components of signatures. (Recall that content-matching *was* considered in the experiments in the previous section.) Since many signatures are identical except for the content-matching part, the default signature set that came with Snort-NG was reduced from a size of 1635 to 305.

Figure 6 shows the effect of increasing the number of rules on the number of automaton states. We can see from the graph that as the number of rules increases, the number of states in `Snort NG` increases much faster than our technique. For 300 rules, Snort-NG automaton contains over 45K states whereas the automaton constructed by our technique has only about 4K states, representing an order of magnitude improvement in space utilization.

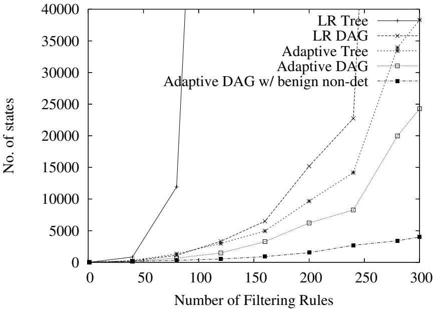


Fig. 7. Effect of Optimizations on Automaton Size for Snort Rules

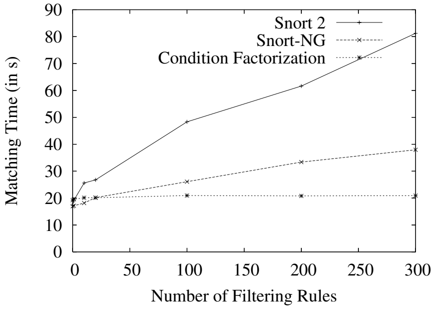


Fig. 8. Matching Time for Packet Classification

Effect of Optimization Techniques. Figure 7 illustrates the effects of different optimizations on the automaton size. We studied different combinations of techniques: with and without sharing of equivalent states in the automata, and with different traversal orders.

- *Order of testing fields.* As compared to left-to-right (L-R) order for examining packet fields, our technique (which uses the *select* function as described in Section 6 produces tree automata that are much smaller: for 120 rules, the L-R automaton had 150,000 states, whereas the tree automaton had less than 3000 states.
- *DAG Vs tree automata.* Our results show that DAG automata were smaller than tree automata by about 25% for our technique. Larger space reductions were achieved with DAG optimization for L-R automata, but still,

L-R automata remain significantly larger than the one constructed by our technique.

- *Benign nondeterminism.* By exploiting benign non-determinism, we were able to achieve dramatic reductions in space usage. This is because Snort contains many rules which test some common fields. Our technique prefers these common fields for testing, since they are the ones that are likely to be discriminating. Once these common fields are tested, the residual rule sets contain many independent subsets.

We point out that a combination of our techniques was necessary to achieve the size reductions we have reported. In particular, benign nondeterminism leads to large improvements in size when combined with discriminating tests. It is much less effective when used with L-R technique, since the factors contributing to the occurrence of independent filter sets do not arise frequently when the L-R technique is used.

7.3 Packet Classification Performance

In this section, we describe experiments to study the runtime performance of packet classification. Unlike Section 7.1, which considered packet classification as well as content-matching time, this section focuses exclusively on the packet classification time in order to measure the raw performance benefits provided by our technique. For these experiments, we used the same 21M packet trace mentioned earlier.

Figure 8 shows the matching time taken by Snort, Snort-NG and our technique for classifying these packets as the number of rules change. In the Figure 8, it can be seen the matching time remains essentially constant with our technique, even as the number of rules are increased from about 10 to 300. In contrast, the matching times for Snort and Snort-NG increase significantly with the number of rules. The base matching time for all the techniques (with no rules enabled) is basically the same, as it corresponds to the time spent by Snort to read the packets from a file and do all related processing except matching.

8 Related Work

[19],[10],[20], [17] are techniques targeted at routers where they can restrict the problem so as to work on a small, predefined set of attributes such as IP address and port. Our focus is on NIDS, where a much larger number of attributes may be tested, and moreover, the tests can be complex.

Pattern matching automata have been extensively studied in the context of term rewriting and theorem proving [15]. Sekar et al [16] presented a technique for adapting the order of examination of fields in order to reduce space and matching time complexity of term-matching automata. Gustafsson and Sagonas [7] extended this technique to handle binary data such as network packets. Our technique generalizes their technique further by adding support for inequalities and disequalties. Moreover, our bit-mask operations are more general than their bit-field operations. More importantly, their automata has an exponential worst-cast space complexity. Although they describe a technique for constructing linear-size *guarded sequential automata*, these automata require runtime operations to manipulate match and candidate sets as a result, their transitions have an $O(N)$ complexity (where N is the number of patterns), while our transitions are $O(1)$ expected time.

Techniques such as BPF [11], DPF [5] and Pathfinder [2] can also be viewed as building matching automata where the packet fields are examined in the order they occur, i.e., they rely on a left-to-right traversal instead of relying on the techniques described in Section 5 for selecting the the tests that are performed. As shown in our evaluation, our techniques result in significant gains in space usage, as compared to a left-to-right traversal.

BPF+ [3] uses global dataflow techniques to identify opportunities for eliminating redundant tests. Our condition factorization technique is more general than those of BPF+, being able to reason about semantic redundancies in the presence of bit-masking operations, and comparisons involving different constants. More importantly, condition factorization takes a step beyond the passive approach of recognizing redundant tests and eliminating them: it proactively decomposes complex tests into more primitive ones so that their common components are exposed and shared.

DPF uses dynamic code generation, which allows dynamic reordering of tests. Dynamic reordering improves performance by detecting match failures earlier. Al-Shaer et al [8] and Gupta et al [6] significantly improve on the dynamic reordering technique used in DPF by using efficient algorithms to maintain statistics regarding the traffic. Their techniques are analogous to profile-based optimizations in compilers, whereas ours is analogous to static-analysis based optimizations. Thus, the two techniques can complement each other.

Vern Paxson [12] developed Bro which is another popular NIDS. Sommer and Paxson [18] enhanced Bro signature matching to use regular expressions. An important difference between Bro and Snort is that Bro is primarily stream-oriented: it assembles packet sequences into streams before applying signatures. Commercial NIDS such as those from CISCO and IBM employ a combination of packet-oriented and stream-oriented matching techniques. The packet classification techniques developed in this paper fit naturally in the context of packet-oriented NIDS (Snort or commercial systems), and can speed them up. Integrating them into a stream-oriented NIDS can be a bit more involved, as these systems may apply certain tests on packet fields against some packets (e.g., the first packet in a stream) but not against others.

9 Conclusions

In this paper we presented new techniques for packet-matching. Our approach is based on the concept of condition factorization, and proactively creates opportunities for sharing common tests across different signatures. Unlike previous techniques, our techniques provide a worst-case polynomial bound on the size of matching automata, while ensuring excellent runtime performance in practice.

Our experimental results demonstrate a 30% gain in end-to-end performance of the popular Snort NIDS due to the use of our techniques. They also demonstrate an order of magnitude reduction in space usage as compared to previous systematic packet classification techniques developed in the context of NIDS, as well as matching times that remain virtually constant as the number of rules is increased.

References

1. Aho, A., Corasick, M.: Efficient string matching: An aid to bibliographic search. Communications of the ACM 18(6), 333–343 (1975)
2. Bailey, M.L., Gopal, B., Pagels, M.A., Peterson, L.L., Sarkar, P.: Pathfinder: A pattern-based packet classifier. In: Operating Systems Design and Implementation, pp. 115–123 (1994)
3. Begel, A., McCanne, S., Graham, S.L.: BPF+: Exploiting global data-flow optimization in a generalized packet filter architecture. In: SIGCOMM, pp. 123–134 (1999)
4. Chandra, S., McCann, P.: Packet types. In: Second Workshop on Compiler Support for Systems Software (WCSSS) (May 1999)
5. Engler, D.R., Kaashoek, M.F.: DPF: Fast, flexible message demultiplexing using dynamic code generation. In: SIGCOMM, pp. 53–59 (1996)
6. Gupta, P., McKeown, N.: Packet classification on multiple fields. In: ACM SIGCOMM (1999)
7. Gustafsson, P., Sagonas, K.: Efficient manipulation of binary data using pattern matching. J. Funct. Program. 16(1), 35–74 (2006)
8. Hazem Hamed, E.A.-S., El-Atawy, A.: On dynamic optimization of packet matching in high-speed firewalls. IEEE Journal on Selected Areas in Communications 24(10) (October 2006)
9. Krügel, C., Tóth, T.: Using decision trees to improve signature-based intrusion detection. In: Vigna, G., Krügel, C., Jonsson, E. (eds.) RAID 2003. LNCS, vol. 2820, pp. 173–191. Springer, Heidelberg (2003)
10. Lakshman, T.V., Stiliadis, D.: High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In: SIGCOMM, pp. 203–214 (1998)
11. McCanne, S., Jacobson, V.: The BSD packet filter: A new architecture for user-level packet capture. In: USENIX Winter, pp. 259–270 (1993)
12. Paxson, V.: Bro: A system for detecting network intruders in real-time. In: USENIX Security (1998)
13. Ramesh, R., Ramakrishnan, I., Warren, D.: Automata-driven indexing of prolog clauses. In: Seventh Annual ACM Symposium on Principles of Programming Languages, San Francisco, pp. 281–290 (1990); Revised version appears in Journal of Logic Programming (May 1995)
14. Roesch, M.: Snort - lightweight intrusion detection for networks. In: 13th Systems Administration Conference, USENIX (1999)
15. Sekar, R., Ramakrishnan, I., Voronkov, A.: Term indexing. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 26, vol. II, pp. 1853–1964. Elsevier Science, Amsterdam (2001)
16. Sekar, R.C., Ramesh, R., Ramakrishnan, I.V.: Adaptive pattern matching. In: Automata, Languages and Programming, pp. 247–260 (1992)
17. Singh, S., Baboescu, F., Varghese, G., Wang, J.: Packet classification using multidimensional cutting. In: SIGCOMM (2003)

18. Sommer, R., Paxson, V.: Enhancing byte-level network intrusion detection signatures with context. In: ACM CCS (2003)
19. Srinivasan, V., Varghese, G., Suri, S., Waldvogel, M.: Fast and scalable layer four switching. In: Proceedings of ACM SIGCOMM 1998, pp. 191–202 (September 1998)
20. Woo, T.Y.C.: A modular approach to packet classification: Algorithms and results. In: INFOCOM (2000)

Choosing NTRUEncrypt Parameters in Light of Combined Lattice Reduction and MITM Approaches

Philip S. Hirschhorn[1], Jeffrey Hoffstein[2], Nick Howgrave-Graham[3], and William Whyte[3]

[1] Wellesley College
[2] Brown University
[3] NTRU Cryptosystems, Inc.

Abstract. We present the new NTRUEncrypt parameter generation algorithm, which is designed to be secure in light of recent attacks that combine lattice reduction and meet-in-the-middle (MITM) techniques. The parameters generated from our algorithm have been submitted to several standard bodies and are presented at the end of the paper.

1 Introduction

Recent research [12] has demonstrated that the NTRUEncrypt parameter sets described in [11] do not provide the claimed level of security (for example, the parameter sets proposed in [11] for 80-bit security only provide about 64 bits of security against the attack of [12])[1]. This paper proposes new parameter sets that are secure against the attacks of [12]. In addition, we present the algorithm used to obtain those parameter sets. Some public key cryptosystems are known to be vulnerable to maliciously chosen parameters [20]; the publication of a parameter generation algorithm greatly reduces the risk that parameters have been "cooked".

The basic framework of NTRUEncrypt allows for great variation in the types of parameter sets generated. For example, polynomials used in the system may be binary, trinary, or "product-form" [8]. This paper focuses on parameter sets for trinary polynomials, as opposed to the binary polynomials of [11], because they generally offer a better trade-off between security and efficiency[2]. We make

[1] The security function used in this paper is simply the processor time required to break an instance of the algorithm. Memory is assumed to be free to the attack, although in fact the attack of [12] requires substantial amounts of memory. This approach is consistent with that recommended by the ASC X9 standard committee.

[2] Previous papers used binary parameters mainly to keep the parameter q at 256 or less, as this made the calculations more suitable for an 8-bit microcontroller; we do not currently believe it is realistic to keep q down to this level, and as such the advantages of trinary polynomials outweigh the advantages of binary polynomials. The "product-form" polynomials of [8] promise to be more efficient even than the trinary polynomials, but the analysis of the security of product-form polynomials against the attack of [12] is considerably more complicated than the analysis of trinary or binary and is not yet complete.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 437–455, 2009.

various other choices to restrict the space that parameters are chosen from. These choices are explained and motivated in Section 1.1.

The original contributions of this paper include:

- Explain in detail the parameter generation algorithm (because its publication helps justify that there are no "backdoors" in the chosen NTRU parameters)
- Explicitly state any underlying assumptions we use to find optimal parameters (because it is important to understand which parts of the NTRU parameters rely on provable statements and which rely on reasonable assumptions)
- Provide an exact calculation of the probability p_s that a correct guess in the meet-in-the-middle stage of the attack will be recognized as such (this was only experimentally calculated in [12])
- Provide exact calculations of various other quantities involved in the hybrid attack when applied to trinary parameters. This includes a mathematically based formula for the probability of a decryption failure with trinary polynomials
- State the standardized NTRU parameters, and their estimated security.

The algorithm has been fully implemented in an object-oriented fashion in C++. The resulting parameter sets are given in Section 7, and a discussion of future issues is given in Section 8.

NTRU parameters are also dependent on the expected running time of lattice reduction algorithms, which can be modeled by extrapolating data achieved in practice. We state the NTRU parameter generation algorithm generically, i.e. the extrapolation line is given as input to the algorithm. When calculating explicit parameter sets we use specific extrapolation lines, which are justified in Appendix A. The accuracy of these extrapolation lines is stated as an explicit assumption, since they have been generated from extensive experimentation with practical lattice reduction techniques. The extrapolation lines in this paper have been obtained using a novel lattice reduction method known as *isodual lattice reduction*, which appears to give considerably better results in practice than previous lattice reduction methods. The isodual lattice reduction method will be the subject of a separate paper.

1.1 NTRU Background

Since we are describing a parameter generation function we must first define what we mean by NTRUEncrypt parameters. Let $\mathcal{R}$ denote the ring $\mathbb{Z}[X]/(X^N - 1, q)$, for some suitable integers q, N. Let $\mathcal{T}_{d_1,d_2}$ denote the set of trinary elements of $\mathcal{R}$ with d_1 entries equal to one, d_2 entries equal to minus one, and the remaining $N - d_1 - d_2$ equal to zero. Let $a \in_R A$ denote the process of picking an element a from the set A uniformly at random.

The NTRUEncrypt parameters, as described in [6], are integers (q, p, N, d_f, d_g, d_r) where the private key is $F \in_R \mathcal{T}_{d_f,d_f}$[3] and the public key is $h = g/f$ in

[3] We take $d_1 = d_2$ here because, for a given value of $d_1 + d_2$, taking $d_1 = d_2$ maximizes the work for an attacker, and because the efficiency depends only on $d_1 + d_2$, not on the individual values of d_1 and d_2.

the ring $\mathcal{R}$ where $g \in_R \mathcal{T}_{d_g+1,d_g}$[4] and $f = 1 + pF$ is invertible for most choices of F. N is necessarily a prime [5], and p is necessarily coprime to q. For any value of q, we denote the factors of q by $\{q_i\}$ and require each q_i to have order $\geq (N-1)/2 \bmod N$ to avoid attacks based on the factorization of h (or other polynomials) in the ring. This eliminates some values of N from consideration. We denote by $\mathcal{P}$ the set of allowable primes for N.

The encryption primitive forms the ciphertext $e = m' + r * h$ in $\mathcal{R}$, where m' and $r \in_R \mathcal{T}_{d_r,d_r}$ are derived from the message m with a randomized padding scheme [9,10,11]. We can write this primitive as a trapdoor one-way function (OWF), which is made more precise in [10,11].

$$OWF(m', r) = m' + r * h$$

The NTRUEncrypt cryptosystem can alternatively be seen as a lattice based cryptosystem [15] in the lattice generated by the row of the following $(2N) \times (2N)$ matrix.

$$\mathcal{L}_{NTRU} = \begin{pmatrix} qI_N & 0_N \\ H & I_N \end{pmatrix},$$

where H denotes a $(N) \times (N)$ circulant matrix generated from h, i.e. $H_{i,j} = h_{i-j \bmod N}$. This lattice is also useful in cryptanalyzing NTRUEncrypt.

For all the parameter sets in this paper we will use the values $q = 2048$, $p = 3$, and $d_r = d_f$, $d_g = \lfloor N/3 \rfloor$[5]. Thus the NTRUEncrypt parameter family we are concerned with can be parametrized by just two parameters: N and d_f.

1.2 Attacks on NTRUEncrypt

At a high level there are two distinct but related attacks known on NTRUEncrypt: an attack based on knowledge of valid (plaintext,ciphertext) pairs which cause decryption failures [10], and an attack based on reversing the one-way function used for key generation and encryption [12].

The notion of decryption failures is slightly uncommon in cryptography, namely that there is a small probability p_{dec} that a valid message can fail to decrypt properly. If an attacker is aware of when this happens in NTRUEncrypt it has been noticed that they can mount an attack on the cryptosystem to recover the private key [10]. Thus in designing parameters for a security level k we ensure that this probability is less than 2^{-k} for a randomly chosen preimage (m', r) of the NTRUEncrypt OWF. The encryption scheme used [11] then provides random (m', r), allowing a proof of security in the random oracle model.

[4] g is unbalanced in this way to increase the chance that g will be invertible in $\mathcal{R}$.

[5] $q = 2048$ is the only value of q for which we have enough lattice reduction experiments to reliably extrapolate lattice reduction times. Further lattice experiments are underway at the moment. Taking q to be a power of 2 allows for efficient reduction $\bmod q$. We take $d_r = d_f$ for convenience, although the result of this is that it is very slightly easier to recover plaintext from ciphertext than private keys from public keys. We take $d_g = \lfloor N/3 \rfloor$ because the thickness of g does not affect efficiency and for security reasons it is sensible to take it to be as thick as possible.

Remaining at a high level, and recalling that our NTRUEncrypt parameter family is parametrized only by N and d_f, the consequence of requiring that decryption failures occur with small probability is that d_f cannot get too large compared to N.

Conversely the attacks based on reversing the one-way function used for key generation and encryption work better when d_f is small compared to N. The challenge facing parameter generation is thus to balance the size of d_f compared to N, whilst keeping N relatively small for efficiency.

The best known attack on reversing the NTRUEncrypt OWF is a combination of the two previously known attacks on NTRUEncrypt, namely Odlykzo's MITM attack, and standard lattice reduction. It is shown in [12] that these can be combined in to one (more efficient) hybrid attack.

The attack consists of two consecutive stages: the first stage is to reduce the initial rows of the public NTRU lattice basis $\mathcal{L}_{NTRU}$ with the best known lattice reduction scheme, and the second stage is to store partial key guesses in boxes dependent on the output of the first stage. The attack is successful if two partial key guesses that collide in the same box can be combined to recover the entire private key.

This paper is written from the perspective of justifying the new parameter generation algorithm which is secure in light of this hybrid attack[6], not from the perspective of explaining the hybrid attack further (the reader is referred to [12] for this). However, to help the reader, informative comments about the attack are embedded in to the paper whenever they are applicable.

2 An Overview of Parameter Generation

2.1 The Criteria for Valid Parameters

As explained in Sections 1.1 and 1.2 the challenge of generating NTRUEncrypt parameters suitable for a security level k is in choosing (N, d_f) such that:

1. The probability of decryption failures, P_{failDec}, is at most 2^{-k} for randomly chosen preimages of the NTRU OWF, *and*
2. The expected work to recover the private key f from the public key h, W_{key}, is at least 2^k, *and*
3. The expected work to recover a plaintext m from the ciphertext c, W_{msg}, is at least 2^k.

There is an implicit assumption here, namely:

Assumption 1. *If decryption failures occur with probability less than 2^{-k} then there is no attack with expected work less than 2^k which exploits the phenomenon of decryption failures.*

One possible line of attack to circumvent Assumption 1 is to find weaknesses in the hash functions such that one can sample preimages (m', r) to the NTRUEncrypt OWF which have decryption failures with a higher probability than

[6] And the decryption failure attacks.

randomly chosen preimages. Whilst such an avenue is theoretically possible it seems a daunting task for an attacker, and would likely require an unfeasibly large number of calls to the decryption oracle. We also note that since the hash functions take the public key as input, an attacker will with high probability need to find a separate set of preimages for each key under attack, preventing them from leveraging a single decryption-failure-causing preimage to attack multiple keys.

We also remark that there has been some work done [16] on linking requirements 2 and 3 above, i.e. in showing that if one has an oracle that can recover NTRUEncrypt messages, then it can be used to recover the NTRUEncrypt private key [16]. Such reductions are typically highly dependent on the structure of the key and OWF preimages so typically do not form tight arguments. In the proposed NTRUEncrypt parameters we have not looked for a provable correspondence between message and key recovery. Instead we independently consider the problem of message recovery in Section 6.

2.2 The Space of Valid Parameters

The space of valid (N, d_f) for $q = 2048$, $p = 3$, $d_r = d_f$, $d_g = \lfloor N/3 \rfloor$ is depicted in the Figure 1.

The curved solid lines denote "k-isopleths"; parameters (N, d_f) for which W_{msg}, the expected work of the hybrid attack is 2^k. The security levels shown corresponds to $k = 112, 128, 192$ and 256 bits of security. Notice that d_f is necessarily at most $\lfloor N/3 \rfloor$ since F is a trinary vector, and this constraint is shown by the (leftmost) straight dashed line. The other constraint on d_f is

$$W_{\text{msg}} \leq P_{\text{decFail}}^{-1} \ .$$

The (central) curved dashed line corresponds to $W_{\text{msg}} = P_{\text{decFail}}^{-1}$. Valid NTRUEncrypt parameters (N, d_f) are below both of these lines.

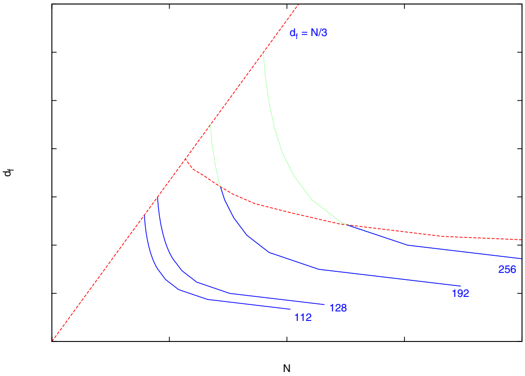


Fig. 1. k-isopleths for d_f vs. N

The exact location of the isopleths of the expected work of the hybrid attack depends on how we are measuring this work. In this report we have two ways of measuring the hybrid cost:

- a *conservative* estimate (which involves being conservative about both lattice running times, and conservative about the expected running time of the MITM component).
- a *current* estimate (which involves more realistic estimates of how quickly the attack can be mounted, with our current knowledge).

Once the isopleths of the expected work of the hybrid attack are fixed, one can apply a "cost" function to each of the (N, d_f) on a k-isopleth and potentially find an "optimal" one. In this report we consider three ways of measuring the cost of a parameter set:

- a *space* metric (trying to optimize the size of the ciphertexts and public key),
- a *speed* metric (trying to optimize the time to perform decryption), and
- a *trade-off* metric (combining the two other metrics).

2.3 The Algorithm

The parameter generation algorithm takes as input a security parameter k and outputs the parameter set corresponding to that security level. At a high level it can simply be described as the following: (a) pick a way to measure cost and a way to measure security, (b) find an initial (N, d_f) on the dashed line which corresponds to a security level of k, and (c) traverse the k-isopleth until the cost metric is optimized. This is made more precise in Algorithm 1.

3 Decryption Failure Probability

The decryption failure probability can be conservatively bounded by the probability that one or more coefficients of $r * g + F * m$ has an absolute value greater than $c = (q-2)/(2p)$.

For trinary polynomials chosen subject to the constraints in this paper, this probability is

$$p_{dec} = N * \texttt{erfc}(c/(\sigma\sqrt{2N})), \tag{1}$$

where $\sigma^2 = 4(d_r + d_f)/(3N)$.

We now justify this statement. In the following calculation we will make the following explicit assumption:

Assumption 2. *The coefficients of r are independent random variables taking the value 1 with probability d_r/N, -1 with probability d_r/N and 0 with probability $(N-2d_r)/N$. We make corresponding assumptions about g, d_g, F, d_f and $m', d_{m'}$.*

Algorithm 1. NTRUEncrypt Parameter Generation

```
1: $i \leftarrow 1$ {The variable $i$ is used to index the set of acceptable primes $\mathcal{P}$}
2: $i^* \leftarrow 0$ {This will become the first index which can achieve the required security}
3: repeat
4:   $N \leftarrow \mathcal{P}_i$
5:   $d_f \leftarrow \lfloor N/3 \rfloor$ {We will try each $d_f$ from $\lfloor N/3 \rfloor$ down to 1}
6:   repeat
7:     $k_1 \leftarrow$ hybridSecurityEstimate$(N, d_f)$ {This is dependent on the chosen security metric}
8:     $k_2 \leftarrow \log_2$(decryptionFailureProb$(N, d_f)$)
9:     if $(k_1 \geq k$ and $k_2 < -k)$ then
10:      $(i^*, d_f^*) \leftarrow (i, d_f)$ {Record the first acceptable index $i$ and the value of $d_f$}
11:    end if
12:    $d_f \leftarrow d_f - 1$
13:  until $(i^* > 0$ or $d_f < 1)$
14:  $i \leftarrow i + 1$
15: until $i^* > 0$
16: $c^* \leftarrow$ cost$(\mathcal{P}_{i^*}, d_f^*)$ {This is dependent on the chosen cost metric}
17: while an increase in $N$ can potentially lower the cost do
18:   $N \leftarrow \mathcal{P}_i$
19:   $d_f \leftarrow d_f^*$ {Note that when $N$ increases the cost must be worse for all $d_f \geq d_f^*$, and that the decryption probability is decreased both by an increase in $N$ and a decrease in $d_f$}
20:   repeat
21:     $k_1 \leftarrow$ hybridSecurityEstimate$(N, d_f)$
22:     $c \leftarrow$ cost$(N, d_f)$
23:     if $(k_1 \geq k$ and $c < c^*)$ then
24:       $(c^*, i^*, d_f^*) \leftarrow (c, i, d_f)$ {Record the improvement in cost and the corresponding $i, d_f$}
25:     end if
26:     $d_f \leftarrow d_f - 1$
27:   until $d_f < 0$
28:   $i \leftarrow i + 1$
29: end while
30: return $(\mathcal{P}_{i^*}, d_f^*)$
```

- Note that line 17 is not specified precisely because it depends on the cost() function used (see Section 4 for a discussion of the possible cost metrics and end-conditions).
- Note that the call to decryptionFailureProb() on line 8 simply uses Equation 1, and the calls to hybridSecurityEstimate() use Algorithm 2.

In order for decryption to succeed, the coefficients of

$$a = p * (r * g + m' * F) + m.$$

must have absolute value less than $q/2$. In fact, it would suffice to know that these coefficients lay in an interval of length less than q in order to do first a translation, then a decryption. Shortly after NTRU was first introduced this second method was viewed as preferential, as it increased the probability of successful decryption. However, theoretically predicting the likelihood of decryption failure by this method was a particularly unwieldy problem, due to the fact that the sizes of the maximum and minimum coefficients were not independent random variables.

The somewhat weaker question of estimating from above the probability

$$p_{dec}(c) = \text{Prob}\,(\text{a given coefficient of } r * g + m' * F \text{ has absolute value } \geq c) \tag{2}$$

can, however, be analyzed quite accurately by a simple application of the central limit theorem, making the assumption described above.

If X_j denotes a coefficient of $r*g+m'*F$, then X_j is a sum of N terms, that is,

$$X_j = \sum_{i=1}^{N} (y_i + z_i)\,,$$

where each $y_i = r_k g_l$ and $z_i = m_s F_t$ for some k, l, s, t. It is then easily checked that

$$\sigma^2 = E(X_j^2) = \sum_{i=1}^{N} \left(E(y_i^2) + E(z_i^2)\right) = \frac{4d_r d_g + 4d_f d_{m'}}{N}.$$

Assuming that N is large, we apply the central limit theorem to X_j, (normalized to have variance equal to 1). Applying the theorem twice, to account for the negative or positive extremes of X_j, we find that

$$\text{Prob}\,(|X_j| \geq C\sigma) < \frac{2}{\sqrt{2\pi}} \int_C^\infty e^{-x^2/2} dx.$$

Writing $c = C\sigma$ we have

$$\text{Prob}\,(|X_j| \geq c) < \frac{2}{\sqrt{2\pi}} \int_{c/\sigma}^\infty e^{-x^2/2} dx.$$

Translating this into the notation of the complementary error function, we get

$$\text{Prob}\,(|X_j| \geq c) < \texttt{erfc}(c/(\sigma\sqrt{2})),$$

with σ as above.

The decryption failure probability can be conservatively bounded by the probability that one or more coefficients of $r*g+F*m$ has an absolute value greater than $c = (q-2)/(2p)$. Thus, repeating the experiment of selecting a coefficient N times, we finally have the probability of decryption failure bounded above by

$$p_{dec}(c) = N\texttt{erfc}(c/(\sigma\sqrt{2})),$$

with $c = (q-2)/(2p)$. For trinary polynomials chosen subject to the constraints in this paper, $\sigma^2 = 4(d_r + d_f)/3$. Experiments with on the order of 10^9 polynomials, with different values for the d's, have shown that this model gives an accurate, while conservative, description of reality.

4 Cost Functions

As explained in Section 2.2 there are three cost metrics of interest:

The space metric. The space metric corresponds to the bit-size of the public key and ciphertexts, i.e. $\text{cost}_{\text{space}} = N \log_2 q$. Since, for constant q, this is independent of d_f, one does not have to traverse the k-isopleth looking to minimize it; it is simply minimized for the least N on the k-isopleth.

The speed metric. The speed metric is derived from the convolution times for a parameter set, i.e. $\text{cost}_{\text{speed}} = Nd_f$.

When trying to minimize this cost notice that an increase in N is offset by a decrease in d_f, thus if N^*, d_f^* corresponds to the current minima on the k-isopleth then one can stop searching the k-isopleth when N increases beyond $N^* d_f^*/(d_f^* - 1)$ without a drop in d_f.

The trade-off metric. In particular environments neither the space metric nor the speed metric may be the natural way to measure cost, but some other metric. We introduce a trade-off metric: $\text{cost}_{\text{trade-off}} = \text{cost}^2_{\text{space}} \times \text{cost}_{\text{speed}}$, which does not have an obvious intrinsic interest, but allows us to show how to handle different metrics.

The criteria for knowing when to stop searching the k-isopleth for this metric can be the same as the speed metric.

5 Hybrid Security

5.1 Overview

The hybrid security function aims to find optimal attack parameters, i.e. parameters such the maximum of lattice security and MITM security is minimized over all attack parameters.

As explained in [12], the hybrid security of a parameter set (N, d_f) is the minimum security over all attack strategies on that parameter set. An attack strategy is defined by the following three values:

- an integer y_2, $N \leq y_2 \leq 2N$ which corresponds to the number of initial rows of the lattice $\mathcal{L}_{NTRU}$ which will be reduced
- a real number α, which roughly corresponds to the quality of the reduced part of the lattice; a larger α corresponds to a better reduced lattice (which takes more time to achieve).
- an integer c which corresponds to the expected number of ± 1's in the last $2N - y_2$ entries of F which we will mount the MITM attack on (note that these entries affect the last $2N - y_2$ rows of $\mathcal{L}_{NTRU}$). By definition, $0 \leq c \leq d_f$.

In performing our analysis we have assumed that the number of $+1$'s and -1's in the last $2N - y_2$ rows is the same. For clarity we make this assumption explicit:

Assumption 3. *An attacker cannot do substantially better by assuming that the number of* 1 *coefficients in the last* $2N - y_2$ *rows is different from the number of* -1 *coefficients.*

A simple way to enumerate the space and hence find the optimal attack strategy is given in Algorithm 2.

Algorithm 2. Optimal Attack Strategy and Hybrid Security Estimation

```
1: for $y_2 = N$ to $2N$ do
2:     for $c = 0$ to $d_f$ do
3:         Find $\alpha$ such that:
           latticeRunningTime$(y_2, \alpha)$ − MITMRunningTime$(y_2, c, \alpha) \approx 0$.
4:         $w \leftarrow \max($latticeRunningTime$(y_2, \alpha)$, MITMRunningTime$(y_2, c, \alpha))$
5:         if ($w^*$ is undefined or $w < w^*$) then
6:             $(w^*, y_2^*, c^*, \alpha^*) \leftarrow (w, y_2, c, \alpha)$ {Record the improved attack strategy, and
               implied attacker work}
7:         end if
8:     end for
9: end for
10: return the optimal attack strategy $(y_2^*, c^*, \alpha^*)$ and the estimated work $w^*$
```

- Notice that the call to latticeRunningTime() is independent of c and simply returns Equation 3. The call to MITMRunningTime() returns $t\mathcal{N}/p_{split}$ where these quantities are defined as in Section 5.3.
- The calculation of α in step 3 can be done by standard root finding techniques.

5.2 Lattice Security

In order to find optimal attack parameters, we need to be able to estimate the work it takes to reduce the first y_2 rows of the lattice $\mathcal{L}_{NTRU}$ with "quality" α.

In Appendix A we discuss what it means for lattice to have "quality" α, and we present the lattice experiments we have conducted to approximate this time. To extrapolate beyond the end of the experimental data we model running time as exponentially dependent on the GSA slope as defined in [19], and cubically dependent on the dimension[7]. This model gives the estimates of running time as 2^w where

$$w = \frac{2m(y_2 - N)}{(1-\alpha)^2} + 3 \log \frac{2(y_2 - N)}{1 - \alpha} + c \tag{3}$$

for some constant m, c.

Extrapolating lattice reduction times is far from a science, but the data seems to support, with reasonable assurance, that practical lattice reduction times exceed the above formula with $m = 0.45$, $c = -110$.

[7] This cubic dependency is heuristic but does not greatly affect the results.

As mentioned in Section 2.2 we allow the parameters sets to be generated with either current security assurances, or conservative security assurances. For the conservative security assurance we use the constants $m = 0.2$, $c = -50$, which hopefully give ample room for improvements in lattice reduction, without requiring that the NTRUEncrypt parameter sets change. We make this assumption explicit below:

Assumption 4. *We assume lattice reduction of the first y_2 elements of the lattice $\mathcal{L}_{NTRU}$ with quality α takes time at least 2^w where w is given by Equation 3, with $m = 0.2$, $c = -50$.*

5.3 MITM Security

In order to find optimal attack parameters we need to be able to estimate the expected work it takes to perform the MITM attack, for a given y_2, α and c.

The following events need to happen for a successful MITM attack:

- The c value must correctly hold how many 1s and -1s are in the last $2N - y_2$ entries of F. The probability of this event is called p_{split}.
- The attacker must enumerate through a large number of guesses of "halves" of the end of F.
- For each one he must apply a CVP reduction algorithm.
- A pair of half-guesses must collide after the CVP algorithm. This probability is referred to as p_s in [12].

We now go through the expected work for each of these events. If a necessary event happens only with a probability p then the expected work is divided by p.

Let $\binom{n}{r_1, r_2}$ denote a "trinomial" quantity, i.e. the number of ways of choosing r_1 positions of one kind, and r_2 positions of another kind in a vector of length n. It is simple to confirm the following relationship between trinomial and binomial quantities: $\binom{n}{r_1, r_2} = \binom{n}{r_1}\binom{n-r_1}{r_2}$.

The probability p_{split}. The probability that the number of 1s and -1s in the last $2N - y_2$ entries of F is equal to c is given by

$$p_{\text{split}} = \binom{y_2 - N}{d_f - c, d_f - c}\binom{2N - y_2}{c, c}\binom{N}{d_f, d_f}^{-1}$$

If the NTRUEncrypt public key was $h = g/F$ then any of the rotations of F would be suitable for the MITM attack, so this probability should be increased to

$$p'_{\text{split}} = 1 - (1 - p_{\text{split}})^N,$$

assuming the N rotations behave like independent trials.

In the case when the NTRUEncrypt public key is $h = g/f$ then we do not see a way to exploit the cyclic symmetry, so when assessing the MITM work in the *current security* case we will be using the probability p_{split}. However, when assessing the MITM work in the *conservative security* case we will assume that the rotations can still somehow be exploited and use the probability p'_{split}.

The probability p_s. In [12] the probability p_s was experimentally calculated, whereas for parameter generation we must calculate this probability mathematically. To make this possible we make the following assumption (which very closely models reality).

Assumption 5. *We assume the orthonormal matrix Y defined in [12] (which is an output of the lattice reduction stage) is suitably random so that the error vector $(g|f_1)Y$ can be modeled by a normal distribution with mean zero, and variance $\sigma^2 = |g|^2 + |f_1|^2 = 2d_g + 2(d_f - c)$.*

As explained in [12] the probability p_s denotes the probability that an “error vector” e is such that $\mathrm{Babai}_B(v) = \mathrm{Babai}_B(v + e)$ where $\mathrm{Babai}_B(v)$ denotes applying Babai's nearest plane CVP algorithm to the point v with respect to the basis B, i.e. the addition of the error vector e does not change the returned close lattice vector.

In NTRUEncrypt the error vector e results from the multiplication of a trinary vector (of norm-squared $2d_g+2(d_f-c)$) with an orthonormal matrix, Y, resulting from lattice reduction. Under the assumption that Y is “suitably random”, we can model the error vector e as (y_2-y_1) coefficients drawn independently from a normal distribution with mean 0 and variance $\sigma^2 = 2d_g + 2(d_f - c)$. This model fits extremely closely with experimental results from 10^9 separate convolution calculations.

Let v be a vector of length $y_2 - y_1$ with coefficients satisfying $0 \le v_i \le q^{\alpha+i(1-\alpha)/(y_2-y_1)}$, i.e. v is weakly reduced with respect to a basis that satisfies the GSA. In this case p_s is the probability that the vector $w = v + e$ satisfies the conditions $0 \le w_i \le q^{\alpha+i(1-\alpha)/(y_2-y_1)}$ for every i. We now show how to calculate p_s exactly, assuming the above model of the error vector e.

Let $D = q^\beta$ for some $\alpha \le \beta \le 1$. We assume any given coefficient is uniform in the range $[0, D)$ and is subject to the addition of an error term drawn from a normal distribution with variance σ^2 and mean 0. For any given $x \in [0, D)$ the probability that this sum is no longer in the range $[0, D)$ – in other words, the probability that a particular coefficient will be reduced incorrectly – is given by:

$$\begin{aligned} f_{D,\sigma}(x) &= \frac{1}{2}\left(\operatorname{erfc}\left(\frac{x}{\sigma\sqrt{2}}\right) + \operatorname{erfc}\left(\frac{D-x}{\sigma\sqrt{2}}\right)\right) \\ &= 1 - \frac{1}{2}\left(\operatorname{erf}\left(\frac{x}{\sigma\sqrt{2}}\right) + \operatorname{erf}\left(\frac{D-x}{\sigma\sqrt{2}}\right)\right) \end{aligned}$$

See Figure 2(a) for an example of this function. Note that, so long as $\sigma^2 < D$, we have $f_{D,\sigma}(0) = 0.5$ and $f_{D,\sigma}(D) = 0.5$ as the values of e_i will be positive or negative with probability 0.5. The expected value of f when x is chosen uniformly from $[0, D)$ is given by:

$$\overline{f_{D,\sigma}} = \frac{1}{D}\int_0^D f_{D,\sigma}(x)dx.$$

This value is also plotted in Figure 2(a).

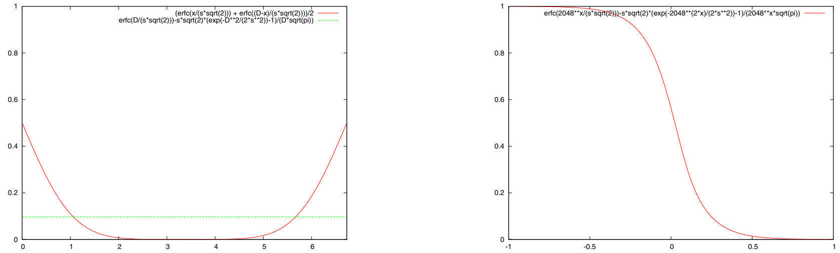


Fig. 2. (a) $f_{D,\sigma}(x)$ and $\overline{f_{D,\sigma}}$ when $D = 2048^{1/4}$, $\sigma^2 = 2/3$, $0 \leq x \leq D$; (b) $\overline{f_{q^\beta,\sigma}}$ when $q = 2048$, $\sigma^2 = 2/3$, $-1 \leq \beta \leq 1$

Now we can analyze the behavior of $\overline{f_{D,\sigma}}$ with D. From well known properties about the error function erf() we know that:

$$\begin{aligned}\int_0^D \operatorname{erf}\left(\frac{x}{\sigma\sqrt{2}}\right) dx &= \sigma\sqrt{2}\int_0^{\frac{D}{\sigma\sqrt{2}}} \operatorname{erf}(y)\, dy \\ &= \sigma\sqrt{2}\left[y \operatorname{erf}(y) + \frac{e^{-y^2}}{\sqrt{\pi}}\right]_0^{\frac{D}{\sigma\sqrt{2}}} \\ &= D \operatorname{erf}\left(\frac{D}{\sigma\sqrt{2}}\right) + \frac{\sigma\sqrt{2}}{\sqrt{\pi}}\left(e^{-\frac{D^2}{2\sigma^2}} - 1\right)\end{aligned}$$

and

$$\int_0^D \operatorname{erf}\left(\frac{D-x}{\sigma\sqrt{2}}\right) dx = \int_0^D \operatorname{erf}\left(\frac{x}{\sigma\sqrt{2}}\right) dx.$$

Thus we have

$$\overline{f_{D,\sigma}} = \operatorname{erfc}\left(\frac{D}{\sigma\sqrt{2}}\right) - \frac{\sigma\sqrt{2}}{D\sqrt{\pi}}\left(e^{-\frac{D^2}{2\sigma^2}} - 1\right).$$

This function is sketched in Figure 2(b) with $D = q^\beta$, $-1 \leq \beta \leq 1$.

Now we can calculate $p_s(y_2, \alpha, q, \sigma)$ which is the probability that none of the coefficients in the middle $y_2 - y_1$ coefficients of γ are reduced incorrectly (note that y_1 is related to y_2 and α by (4)):

$$\begin{aligned}p_s &= \left(1 - \frac{2}{3q}\right)^{y_1} \prod_{i=0}^{y_2 - y_1}\left(1 - \overline{f_{q^{\frac{\alpha(y_2-y_1)+i(1-\alpha)}{y_2-y_1}},\sigma}}\right) \\ &= \left(1 - \frac{2}{3q}\right)^{\frac{2N - y_2(1+\alpha)}{1-\alpha}} \prod_{i=0}^{\frac{2(y_2-N)}{1-\alpha}}\left(1 - \overline{f_{q^{\frac{2\alpha(y_2-N)+i(1-\alpha)^2}{2(y_2-N)}},\sigma}}\right)\end{aligned}$$

The time for the CVP algorithm. Each step in the MITM search phase involves a reduction against a $(y_2) \times (y_2)$ matrix. In this paper we assume the reduction is performed by Babai's method [1]. Babai's reduction is experimentally found to take about $t = y_2^2/2^{1.06}$ operations where bit operations are defined as in [14].

For conservative estimates, we assume that since the MITM phase involves multiple reductions against the same reduced matrix, there will be some optimization involving precomputation that reduces the running time by a factor of y_2 to $t' = y_2/2^{1.06}$.

We note that there may be other means of carrying out this weak-CVP reduction phase. Babai's reduction is likely to be the most efficient when measured purely in terms of the time for the reduction. However, it is conceivable that a slower reduction algorithm exists which results in higher values for p_s, and that this algorithm might give better results for an attacker than the current approach.

Assumption 6. *We assume that Babai's nearest plane algorithm is the most effective CVP approach to use in the hybrid attack (in terms of collision probability divided by time).*

We note that contradicting this assumption does not necessarily mean that the proposed NTRUEncrypt parameters are weak, since conservative security estimates have been made in many other areas. However it is an interesting open question to know whether or not there is a more efficient CVP approach to be used in the hybrid attack.

The expected number of half-guesses required. It is explained in [12] that the hybrid attack works by choosing a linear combination of the last $2N - y_2$ rows of $\mathcal{L}_{NTRU}$ with $c/2$ 1s and $c/2$ -1s. The number of such combinations is given by

$$\mathcal{N}_0 = \binom{2N - y_2}{c/2, c/2}\binom{c}{c/2, c/2}^{-1}.$$

However since we require the two halves to collide after the CVP stage the probability of picking such a "good" combination is only $p_s\mathcal{N}_0^{-1}$.

Moreover in [12] it is argued that, by the birthday paradox, one should expect to try $\sqrt{p_s\binom{c}{c/2,c/2}}$ samples before finding two halves that actually match. Thus the number of trials before a collision will occur for the *current security* is approximated by

$$\mathcal{N} = \binom{2N - y_2}{c/2, c/2}\left(\binom{c}{c/2, c/2}p_s\right)^{-1/2}$$

In estimating the number of trials before collision with occur with *conservative security* we assume that generalized birthday methods can be applied so just one[8] "good" combination is sufficient to find a collision, i.e. we use the estimate $\mathcal{N}' = \mathcal{N}_0/p_s$. This conservative assumption may be close to reality as indicated by the work in [13].

[8] Or more realistically a small constant number of good combinations.

6 Message Recovery

One can estimate the hardness of message recovery as opposed to key recovery in a similar way to the above. The ciphertext point $(c, 0)$ is only $(m', -r)$ away from a lattice point of $\mathcal{L}_{NTRU}$. In our family of NTRU parameters we know $d_r = d_f$, but m' is a random trinary vector, with an un-fixed number of 0s, 1s and -1s. If m' is very sparse for example, then message recovery could be easier than key recovery. We also note that the value of $m'(1)$ is leaked by each ciphertext since $c(1) = m'(1) + r(1)h(1) \bmod q$.

Clearly an encrypter cannot leak the private key in a public-key encryption scheme (since they know no more information than an attacker), but they might leak their message m. However we observe that the encrypter does see m', r, so she can re-encrypt if she wants. We note that the probability of re-encryption depends on the parameter set, and parameters may be rejected if this probability is too high for comfort.

In this report we make the following simplifying assumption:

Assumption 7. *An encrypter that re-encrypts whenever the number of* 1*s or* -1*s in* m' *falls below* d_f *does not make message recovery fall below the required security level.*

When comparing the message recovery problem to the key recovery problem there are two factors to take in to consideration: one that is good for an attacker, and one that is bad. Firstly if an attacker knows m' only has d_f 1s and -1s, the message recovery lattice problem is actually slightly easier than the key recovery problem since p_s will be larger in this case (due to the fact that g is thicker than m'). However the second factor is that m' is unknown to an attacker, so the probability of a sparse m' should be factored in to an attackers strategy. We note that the probability of a random trinary N-vector having d_m ones and d_m minus ones is given by $\binom{N}{d_m, d_m}$.

To garner belief in Assumption 7 we compared the expected p_s for message security to the p_s for key security (in the case of ciphertexts satisfying $m'(1) = 0$) for each of the optimal attack parameters given in Section 7. The expected p_s probability for the message recovery problem was improved by at most 0.1 bits from the key recovery p_s probability, which suggests that message recovery is very closely tied to key security, and hence Assumption 7 is realistic.

7 Parameter Sets

The results of running the parameter generation algorithm under conservative assumptions about the strength of lattice attacks and MITM attacks is given in Table 1. For each security level k we give the optimized parameter sets (N, d_f) for each of the three cost metrics.

We also assess the strength, k', of parameter sets under current assumptions about the strength of lattice attacks, and MITM attacks, to show the safety margin "built-in" to the parameters.

Table 1. Standardized NTRU Parameters (conservative)

	space	trade-off	speed
k	$(N, d_f), k'$	$(N, d_f), k'$	$(N, d_f), k'$
112	$(401, 113), 154.88$	$(541, 49), 141.766$	$(659, 38), 137.861$
128	$(449, 134), 179.899$	$(613, 55), 162.385$	$(761, 42), 157.191$
192	$(677, 157), 269.93$	$(887, 81), 245.126$	$(1087, 63), 236.586$
256	$(1087, 120), 334.85$	$(1171, 106), 327.881$	$(1499, 79), 312.949$

Table 2. Optimal attack parameters

	space	trade-off	speed
k	$(N, d_f), (y_2, c, \alpha)$	$(N, d_f), (y_2, c, \alpha)$	$(N, d_f), (y_2, c, \alpha)$
	$(y_1, p_s, p_{split}, Y, t)$	$(y_1, p_s, p_{split}, Y, t)$	$(y_1, p_s, p_{split}, Y, t)$
112	$(401, 113), (693, 27, 0.095)$	$(541, 49), (800, 15, 0.149)$	$(659, 38), (902, 13, 0.175)$
	$(48, -45.4, -0.6, 57.7, 8.4)$	$(192, -26.9, -13.1, 63.6, 8.6)$	$(313, -21.9, -17.7, 63.7, 8.8)$
128	$(449, 134), (770, 35, 0.100)$	$(613, 55), (905, 17, 0.142)$	$(761, 42), (1026, 15, 0.183)$
	$(57, -49.0, -0.3, 70.2, 8.5)$	$(225, -31.5, -14.9, 72.9, 8.8)$	$(378, -23.1, -20.9, 75.2, 8.9)$
192	$(677, 157), (1129, 45, 0.096)$	$(887, 81), (1294, 27, 0.143)$	$(1087, 63), (1464, 23, 0.175)$
	$(129, -67.4, -2.0, 113.6, 9.1)$	$(345, -43.9, -21.9, 117.0, 9.3)$	$(550, -34.2, -31.9, 116.7, 9.5)$
256	$(1087, 120), (1630, 39, 0.127)$	$(1171, 106), (1693, 37, 0.144)$	$(1499, 79), (1984, 29, 0.174)$
	$(386, -64.1, -24.9, 157.9, 9.6)$	$(474, -56.0, -28.7, 161.8, 9.7)$	$(809, -44.4, -47.8, 153.9, 9.9)$

For the reader that is interested in knowing the exact attack parameters which match lattice security and MITM security (and hence give the security level), we give this information in Table 2. In this table Y denotes the number of iterations of the MITM attack, and t denotes the bit-security per iteration (i.e. the cost of Babai's CVP algorithm).

8 Conclusions

We have introduced a family of NTRU parameters parametrized by (N, d_f) and shown how to estimate their strength. We have used this algorithm to generate parameters sets for the $k = 112$, 128, 192 and 256-bit security levels under conservative assumptions about the effectiveness of lattice reduction algorithms and the MITM attack. To highlight the safety margin we have built in to the new parameters we have also estimated how hard the parameters are to attack under more current assumptions.

A future line of work may be in expanding the size of the NTRU family of parameters to allow for even better optimized parameters. For example the family we have defined uses a fixed $q = 2048$, whereas a smaller q could reduce the bandwidth and public key-size used in the cryptosystem. To propose such parameters more lattice experiments would have to be run at lower values of q, and one would need to ensure that the decryption failure probability is still small enough.

References

1. Babai, L.: On Lovasz' lattice reduction and the nearest lattice point problem. Combinatorica 6(1), 1–13 (1986)
2. Cavallar, S., Dodson, B., Lenstra, A.K., Lioen, W., Montgomery, P.L., Murphy, B., te Riele, H.J.J., et al.: Factorization of a 512-bit RSA modulus. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 1–17. Springer, Heidelberg (2000)
3. Coppersmith, D., Shamir, A.: Lattice Attack on NTRU. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 52–61. Springer, Heidelberg (1997)
4. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer, Heidelberg (2008)
5. Gentry, C.: Key recovery and message attacks on NTRU-composite. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, p. 182. Springer, Heidelberg (2001)
6. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A new high speed public key cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998)
7. Hoffstein, J., Silverman, J.H.: Invertibility in truncated polynomial rings. Technical report, NTRU Cryptosystems, Report #009, version 1 (October 1998), http://www.ntru.com
8. Hoffstein, J., Silverman, J.H.: Random small hamming weight products with applications to cryptography. Discrete Applied Mathematics 130(1), 37–49 (2003)
9. Howgrave-Graham, N., Nguyen, P., Pointcheval, D., Proos, J., Silverman, J.H., Singer, A., Whyte, W.: The Impact of Decryption Failures on the Security of NTRU Encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 226–246. Springer, Heidelberg (2003)
10. Howgrave-Graham, N., Silverman, J.H., Singer, A., Whyte, W.: NAEP: Provable Security in the Presence of Decryption Failures IACR ePrint Archive, Report 2003-172, http://eprint.iacr.org/2003/172/
11. Howgrave-Graham, N., Silverman, J.H., Whyte, W.: Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3 CT-RSA, pp. 118–135 (2005)
12. Howgrave-Graham, N.: A hybrid meet-in-the-middle and lattice reduction attack on NTRU. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 150–169. Springer, Heidelberg (2007)
13. Joux, A., Howgrave-Graham, N.: Generalized birthday problems applied to subset sum (manuscript)
14. Lenstra, A., Verheul, E.: Selecting Cryptographic Key Sizes. Journal of Cryptology 14(4), 255–293 (2001)
15. Micciancio, D.: Improving Lattice Based Cryptosystems Using the Hermite Normal Form. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 126–145. Springer, Heidelberg (2001)
16. Mol, P., Yung, M.: Recovering NTRU Secret Key from Inversion Oracles. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 18–36. Springer, Heidelberg (2008)
17. Rivest, R., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21, 120–126 (1978)
18. RSA Laboratories, RSAES-OAEP Encryption Scheme, ftp://ftp.rsasecurity.com/pub/rsalabs/rsa/_algorithm/rsa-oaep_spec.pdf
19. Schnorr, C.P.: Lattice Reduction by Random Sampling and Birthday Methods. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 145–156. Springer, Heidelberg (2003)

20. Vaudenay, S.: Hidden Collisions on DSS. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 83–88. Springer, Heidelberg (1996)

A Lattice Experiments

In this section we aim to give plausible estimates for the running time to reduce the first y_2 rows of the NTRU lattice $\mathcal{L}_{NTRU}$ with "quality" α.

It would be nice if there were standard ways to do this is the academic literature, but unfortunately no such work has been done. The paper [4] sounds promising but, despite the title, it does not allow us to predict lattice reduction times for a given quality of reduced basis if we, say, had computing power equivalent to 2^{80} operations.

We note that our notion of the reduction quality α is not strictly a traditional notion[9] of the quality of reduction of a lattice, but it is the most natural measure for the MITM attack.

We define the *profile* of an NTRU basis $\{b_1, b_2, \ldots, b_{2N}\}$ to be a plot of the $\log_q |b_i^*|$. As in [12] we assume that the profile after lattice reduction has taken place looks like Figure 3, that is there is an initial flat portion, followed by an almost linear reduction in the $|b_i^*|$, as predicted by the Geometric Series Assumption (GSA) first stated in [19].

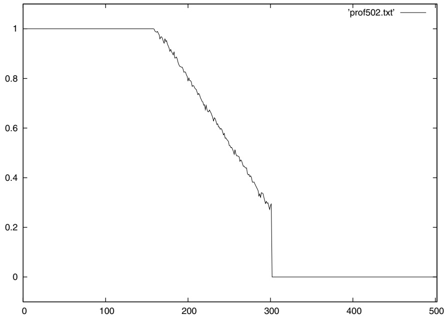


Fig. 3. $\log_{197} |b_i^*|$ for $i = 1, \ldots, 502$, with $y_2 = 302$

The final $2N - y_2$ rows are shown in Figure 3 are not touched by lattice reduction (so the $\log_q |b_i^*|$ remain 0). We define y_1 to be the number of the initial "reduced" vectors satisfying $\log_q |b_i^*| = 1$. These initial y_1 vectors are also not really touched by lattice reduction, so we look on this as a $y_2 - y_1$ dimensional lattice problem. Indeed, as explained in [12], the reduced basis may be produced by extracting and reducing a $(y_2 - y_1) \times (y_2 - y_1)$ submatrix from $\mathcal{L}_{NTRU}$, and then applying some simple matrix operations.

We define $\alpha = |b_{y_2}^*|$ to be the size of the last b_i^* in the reduced portion, and we note that spending more time on lattice reduction will increase this value.

[9] Traditional notions typically involve the ratio of the smallest vector found to the smallest vector in the lattice, but they are closely related to our notion since such bounds generally come from bounding the last $|b_i^*|$ achieved by lattice reduction.

Note that, by the invariance of the determinant, and assuming the GSA, all such profiles satisfy $y_1 + (1/2)(y_2 - y_1)(1 + \alpha) = N$, so we have:

$$y_1 = \frac{2N - y_2(1+\alpha)}{1-\alpha}, \tag{4}$$

which implies the dimension of the lattice problem is $n = y_2 - y_1 = 2(y_2 - N)/(1 - \alpha)$.

We define the "GSA-slope" δ to be the average of $\delta_i = \log |b_i^*| - \log |b_{i+1}^*|$ for $y_1 < i \leq y_2$. Assuming the GSA we have $\delta = (1 - \alpha)^2/(2(y_2 - N))$.

A reasonable way to model[10] lattice reduction running times is to expect a small polynomial dependency on the lattice dimension (i.e. n^3), and a singly exponential[11] dependency on δ^{-1}.

As part of research for this paper we have developed a new and very effective lattice reduction technique, which we refer to as "isodual lattice reduction". We will describe this technique more fully in a separate paper. The effectiveness of this technique is shown by the fact that we have achieved reduced bases in time about 2^{43} which if one had simply increased the BKZ blocksize would have needed about 2^{100} work. The isodual running times are broadly in line with the model of running times presented in this paper.

We suggest two levels of the asymptotic hardness of lattice reduction on NTRU lattices with $q = 2048$. One generated by extrapolating the (end of the) best known running times in low dimension (Equation 5), and the other (Equation 6) is proposed as a more aggressive challenge for lattice reduction:

$$\log_2 t = 0.2\delta^{-1} + 3\log n - 50 = 0.4\frac{y_2 - N}{(1-\alpha)^2} + 3\log\frac{2(y_2 - N)}{1-\alpha} - 50, \text{ or} \tag{5}$$

$$\log_2 t = 0.45\delta^{-1} + 3\log n - 110 = 0.9\frac{y_2 - N}{(1-\alpha)^2} + 3\log\frac{2(y_2 - N)}{1-\alpha} - 110. \tag{6}$$

Figure 4 shows the number of initial q-vectors which were removed for various lattice strategies, and the asymptotes with $\alpha = 0$.

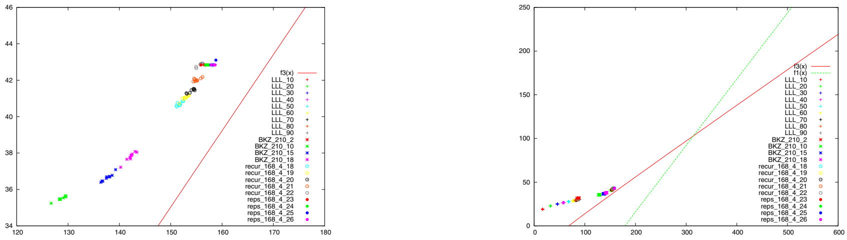


Fig. 4. Best known lattice running times

[10] The veracity of this model should be examined further, especially when $\alpha > 0$, but it does seem a reasonable model.

[11] It can be no "more secure" that this asymptotically since an NTRU-lattice can be fully searched in time q^N.

Broadcast Attacks against Lattice-Based Cryptosystems*

Thomas Plantard and Willy Susilo

Centre for Computer and Information Security Research
School of Computer Science and Software Engineering
University of Wollongong, Australia
{thomaspl,wsusilo}@uow.edu.au

Abstract. In 1988, Håstad proposed the classical broadcast attack against public key cryptosystems. The scenario of a broadcast attack is as follows. A single message is encrypted by the sender directed for several recipients who have different public keys. By observing the ciphertexts only, an attacker can derive the plaintext without requiring any knowledge of any recipient's secret key. Håstad's attack was demonstrated on the RSA algorithm, where low exponents are used. In this paper, we consider the broadcast attack in the lattice-based cryptography, which interestingly has never been studied in the literature. We present a general method to rewrite lattice problems that have the same solution in one unique easier problem. Our method is obtained by intersecting lattices to gather the required knowledge. These problems are used in lattice based cryptography and to model attack on knapsack cryptosystems. In this work, we are able to present some attacks against both lattice and knapsack cryptosystems. Our attacks are heuristics. Nonetheless, these attacks are practical and extremely efficient. Interestingly, the merit of our attacks is not achieved by exploring the weakness of the trapdoor as usually studied in the literature, but we merely concentrate on the problem itself. As a result, our attacks have many security implications on most of the lattice-based or knapsack cryptosystems.

Keywords: Broadcast attack, lattice-based cryptosystem, knapsack cryptosystem, intersecting lattice.

1 Introduction

In 1988, Håstad [1] proposed the first broadcast attack against public key cryptosystems. The attack enables an attacker to recover the plaintext sent by a sender to multiple recipients, without requiring any knowledge of the recipient's secret key. Håstad's attack was originally proposed against the RSA public key cryptosystem that incorporates low exponents. To prevent this classical attack, several researchers have studied the necessity to have a strong security notion in the single user setting in contrast to the multi user setting [2,3]. For instance,

* This work is supported by ARC Discovery Grant DP0663306.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 456–472, 2009.

it is well known that to avoid such an attack, paddings (in the random oracle model) will need to be incorporated to achieve the IND-CCA security notion [2,3]. Nevertheless, this type of attacks has never been discussed in the lattice-based scenario.

Our Contribution. In this paper, we revisit the classical broadcast attack and consider it in the lattice-based scenario. Interestingly, this is the first work that considers this type of attack in lattice-based scenario. Our approach is as follows. We present a general method to rewrite lattice problems that have the same solution in one unique easier problem. Our method is obtained by intersecting lattices to gather the required knowledge. These problems are used in lattice based cryptography and to model attack on knapsack cryptosystems. We are able to present some attacks against both lattice and knapsack cryptosystems. Our attacks are heuristics. Nonetheless, these attacks are practical and extremely efficient, as demonstrated in our experiment. We also discuss some countermeasures against such attacks in the context of lattice-based cryptography.

1.1 Related Works

Knapsack Cryptosystems

In 1978 [4], Merkle and Hellman proposed the first public key cryptosystem based on a NP-hard problem, namely the knapsack problem. This is the first practical public key cryptosystem as a positive answer to the proposed seminal notion of public key cryptography by Diffie and Hellman [5]. The knapsack problem is as follows.

Problem 1 (Knapsack). Let $a_1, \ldots, a_n \in \mathbb{N}$ n positive integers and $s \in \mathbb{N}$ a positive integer. The *Knapsack Problem* is to find, if there exists, $\alpha_i \in \{0,1\}$, $i = 1, \cdots n$, (n Boolean) such that

$$\sum_{i=1}^{n} \alpha_i a_i = s.$$

The problem to find whether there exists such α_i is called the Knapsack Decision Problem.

Theorem 1 (Karp [6]). *The Knapsack Decision Problem is NP-Complete.*

The cryptosystem proposed in [4], and most of other knapsack cryptosystems, can been illustrated as follows.

- **Setup:** Create n integers $a_1, \ldots, a_n$ with a trapdoor function to solve the Knapsack Problem on a_i. Provide $a_1, \ldots, a_n$ as public.
- **Encrypt:** To encrypt a message $m \in [0,1]^n$, compute

  $$s = \sum_{i=1}^{n} m_i a_i.$$

  Publish s as the encrypted message of m.

- **Decrypt:** Use the trapdoor to solve the knapsack problem on $a_1, \ldots, a_n$ and s and extract m.

Merkle-Hellman's first proposition was attacked severely and broken using two different methods: the first attack on the trapdoor itself was proposed by Shamir [7,8] and the second attack on the knapsack problem using lattice theory was proposed by Adleman [9]. In 1985 [10], Lagarias and Odlyzko proposed a general attack against knapsack cryptosystems. Their attack do not incorporate the weakness on the trapdoor itself, rather than only using the fact that the knapsack problems produced are generally weaker that a random one. This weakness appears in a lower density of the knapsack problem. The density of a knapsack problem is defined as

$$d = \frac{n}{max_{i=1}^{n} \log_2 a_i}.$$

Density represents a trade-off between the need to be able to decrypt (and hence, to have a unique solution) using a low density and an acceptable security level using a bigger density. A lot of improvements have be made in order to attack lower density knapsack [11,12,13,14]. For instance, in [12], the authors successfully cryptanalyzed knapsack cryptosystems with density less than 0.9408. These low density attacks use lattice reduction tools. However, some improvements of knapsack cryptosystems were also proposed (e.g. [15,16]) with a bigger density, generally close to 1. We refer the reader to [17] for these two faces of knapsack cryptology. Nonetheless, as mentioned in [18], the knapsack cryptosystem proposed by Okamoto, Tanaka and Uchiyama in 2000 [16] seems to be the only remaining secure knapsack cryptosystem.

Lattice-based cryptosystems

In 1997, Ajtai and Dwork [19] proposed the first lattice cryptosystem where its security is based on a variant of the Shortest Vector Problem (SVP). This cryptosystem received wide attention due to a surprising security proof based on worst-case assumptions. Nonetheless, this cryptosystem is merely a theoretical proposition and it cannot be used in practice. Furthermore, Nguyen and Stern presented a heuristic attack against this cryptosystem [20]. Until then, this initial proposition has been improved [21,22,23] and inspiring for other cryptosystems based on SVP [24,25,26]. The main drawback in these cryptosystems is a huge extension factor between the initial message and its encrypted version.

In 1996, Goldreich, Goldwasser and Halevi (GGH) [27] proposed an efficient way to use lattice theory to build a cryptosystem inspired by McEliece cryptosystem [28] and based on the Closest Vector Problem (CVP). Their practical proposition of a cryptosystem was attacked and broken severely by Nguyen in 1999 [29]. However, the general idea is still viable. Until then, the other propositions were made using the same principle [30,31,32].

In the following, we briefly review the GGH cryptosystem. A GGH cryptosystem comprises of the following algorithms.

- **Setup:** Compute a "good basis" A and a "bad basis" B of a lattice $\mathcal{L}$,
$$\mathcal{L}(A) = \mathcal{L}(B).$$
Provide B as public and keep A secret.
- **Encrypt:** To encrypt a vector-message m: use the bad basis to create a random vector r of $\mathcal{L}$. Publish the encrypted message which is the addition of the vector message with the random vector:
$$c = m + r.$$
- **Decrypt:** Use the good basis to find the closest vector in the lattice of the encrypted message c. The closest vector of the encrypted message c is the random vector r[1]. Subtract the random vector of the encrypted message to obtain the vector message m.

Remark 1. In their initial paper [27], Goldreich, Goldwasser and Halevi also proposed another cryptosystem where the message is transformed into a lattice point prior to adding to it a random vector noise.

The important points for the security and efficiency of those cryptosystems are defined as follows.

i) It is easy to compute a "bad basis" from a "good basis", but it is difficult to compute a "good basis" from a "bad basis".
ii) It is easy to create a random vector of a lattice even with a "bad basis".
iii) It is easy to find the closest vector with a "good basis" but difficult to do so with a "bad basis".

After the first Nguyen's attack [29], utilization of the initial GGH proposition requires lattice with big dimension (> 500), to ensure its security. Nonetheless, the computation of the closest vector even with a "good basis" becomes very expensive. In 2000, Fischlin and Seifert [30] proposed a very intuitive way to build lattice with good basis which are able to solve the closest vector problem. They used a tensor product of lattice to obtain a divide and conquer approach to solve CVP. In 2001, Micciancio [31] proposed some major improvements of the speed and the security of GGH. In this scheme, the public key uses a Hermite Normal Form (HNF) for the bad basis. The HNF basis is better to answer the inclusion question and it also seems to be more difficult to transform to a "good basis" compared to another basis. In 2003, Paeng, Jung and Ha [32] proposed to use some lattices build on polynomial ring. However, in 2007, Han, Kim, and Yeom [33] used lattice reduction to cryptanalysis this scheme. Their attack can successfully recover the secret key even in a huge dimension (> 1000) and make the PJH scheme unusable. However, there exists a secure (and yet 'unbroken') cryptosystem using polynomial representation, namely the NTRU cryptosystem, for N^{th} degree truncated polynomial ring units. NTRU was originally proposed in 1998 by Hoffstein, Pipher and Silverman [34]. Even if this cryptosystem was not modelled initially as a GGH-type cryptosystem, it can actually be represented as one. This has been useful specially for analysing its security [35].

[1] Under the supposition that the norm of m is sufficiently small.

Organization of the Paper

The rest of this paper is organized as follows. In the next section, we recall some basic concepts of lattice theory. Section 3 presents the main theorem which is how to intersect lattices to simplify lattice problems. Some practical attacks inspired by this main theorem are proposed in Section 4. Section 5 presents some test results. We conclude the paper in Section 6 by presenting some solutions against these new broadcast attacks.

2 Lattice Theory

In this section, we will review some concepts of the lattice theory useful for the comprehension of this paper. For a more complex account, we refer the readers to [36].

The lattice theory, also known as the geometry of numbers, has been introduced by Minkowski in 1896 [37]. A complete discussion on the basic of lattice theory can be found from [38,39,40].

Definition 1 (Lattice). *A lattice $\mathcal{L}$ is a discrete sub-group of $\mathbb{R}^n$, or equivalently the set of all the integral combinations of $d \leq n$ linearly independent vectors over $\mathbb{R}$.*

$$\mathcal{L} = \mathbb{Z}\, b_1 + \cdots + \mathbb{Z}\, b_d, \quad b_i \in \mathbb{R}^n.$$

$B = (b_1, ..., b_d)$ is called a basis of $\mathcal{L}$ and d, the dimension of $\mathcal{L}$, noted dim($\mathcal{L}$).
We will refer $\mathcal{L}(B)$ as a lattice of basis B.
We will represent a lattice basis by a matrix $B \in \mathbb{R}^{d,n}$ where each rows $B[i]$ of B correspond to a vector b_i of the basis.

Theorem 2 (Determinant). *Let $\mathcal{L}$ a lattice. There exists a real value, denoted as* $\det \mathcal{L}$*, such that for any basis B, we have*

$$\det \mathcal{L} = \sqrt{\det\left(BB^T\right)}.$$

$\det \mathcal{L}$ *is called the determinant of $\mathcal{L}$.*

For a given lattice $\mathcal{L}$, there exists an infinity of basis. However, the Hermite Normal Form basis (Definition 2) is unique [41].

Definition 2 (HNF). *Let $\mathcal{L}$ a integer lattice of dimension d and $H \in \mathbb{Z}^{d,n}$ a basis of $\mathcal{L}$. H is a Hermite Normal Form basis of $\mathcal{L}$ if and only if*

$$\forall 1 \leq i,j \leq d \quad H_{i,j} \begin{cases} = 0 & \textit{if } i > j \\ \geq 0 & \textit{if } i \leq j \\ < H_{j,j} & \textit{if } i < j \end{cases}$$

The HNF basis can be computed from a given basis in a polynomial time [42]. For efficient solutions, we refer the readers to [43].

Remark 2. As it was remarked by [31], the HNF basis is a "good basis" for solving the problem of inclusion of a vector in a lattice [41] or more generally finding a basis of a lattice from a set of non-independent vectors generating this lattice [36].

Intersecting lattice is the main tool used in this paper. Lattices intersection can easily be done using dual lattices (Definition 3).

Definition 3 (Dual). *Let $\mathcal{L}$ a lattice and B a basis of $\mathcal{L}$. Then, $\mathcal{L}^*$ is noted as the dual lattice of $\mathcal{L}$ and $(BB^T)^{-1}B$ is a basis of $\mathcal{L}^*$*[2].

Property 1 (Intersection). Let $\mathcal{L}_1, \mathcal{L}_2$ two lattices. Then,

$$\mathcal{L}_1 \cap \mathcal{L}_2 = (\mathcal{L}_1^* \cup \mathcal{L}_2^*)^* .$$

As $\mathcal{L}_1 \subseteq \mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$, $\mathcal{L}_1$ is called a sublattice of $\mathcal{L}$.

Remark 3 (Union). The lattice union of two lattices is generated by the set of vectors composed by the union of the two sets of vector of each basis. As remark before (Remark 2), using HNF for example, we can build from this set of non-independent vector, a basis.

The lattice theory problem is based on distance minimization. The natural norm used in lattice theory is the euclidean norm.

Definition 4 (Euclidean norm). *Let w a vector of $\mathbb{R}^n$. The euclidean norm is the function $\|.\|$ defined by*

$$\begin{aligned}\|w\| &= \sqrt{< w, w >} \\ &= \sqrt{ww^T} \\ &= \sqrt{\sum_{i=1}^{n} w_i^2}\end{aligned}$$

Using a norm, we can define some other invariants crucial in lattice theory.

Definition 5 (Successive Minima). *Let $\mathcal{L}$ a lattice and $i \in \mathbb{N}$ an integer. The i^{th} Successive Minima, noted $\lambda_i(\mathcal{L})$ is the smallest real number such there exist i non-zero linear independent vector $v_1, \ldots, v_i \in \mathcal{L}$ with*

$$\|v_1\|, \ldots, \|v_i\| \leq \lambda_i(\mathcal{L}).$$

The problem to find such a vector v_1 is called the Shortest Vector Problem (SVP).

Theorem 3 (Ajtai [44]). *SVP is NP-Hard under randomized reductions.*

Another important invariant is the Hermite invariant which is defined as follows.

[2] This definition of the duality is extremely practical and doesn't represent the full interest of this notion. However, we will only focus on notion needed for the understanding of this paper.

Definition 6 (Hermite Invariant). *Let $\mathcal{L}$ a lattice. The Hermite invariant, denoted as $\gamma(\mathcal{L})$, is the real number such that*

$$\gamma(\mathcal{L}) = \left(\frac{\lambda_1(\mathcal{L})}{\det(\mathcal{L})^{1/dim(\mathcal{L})}} \right)^2 .$$

There exist two extremely useful properties around this invariant.

Theorem 4 (Minkowski [37]). *For any lattice $\mathcal{L}$ of dimension d,*

$$\gamma(\mathcal{L}) \leq 1 + \frac{d}{4}.$$

The second theorem provides a general property which concerns random lattices.

Theorem 5 (Ajtai [45]). *Let $\mathcal{L}$ a random lattice of dimension d. Then,*

$$\frac{\lambda_i(\mathcal{L})}{\det(\mathcal{L})^{1/d}} \simeq \sqrt{\frac{d}{2\pi e}}.$$

Corollary 1. *Let $\mathcal{L}$ a random lattice of dimension d. Then,*

$$\gamma(\mathcal{L}) \simeq \frac{d}{2\pi e}.$$

Random lattice is a complex notion [46,47,45]. Goldstein and Mayer's characterization of random lattices [47] allows to create random lattices for experiment for example [48]. We will use the same method in our practical section (Section 5) to evaluate our method in the case of random lattices.

Remark 4. Hermite invariant is a way to evaluate the weakness of a lattice. If the value is smaller than the average $\frac{d}{2\pi e}$ on a lattice, then it will be "easier" to solve SVP or other related problem on it.

Another useful invariant is the lattice gap defined in [49] for practical reason.

Definition 7 (Lattice Gap). *Let $\mathcal{L}$ a lattice. The gap, noted $\alpha(\mathcal{L})$, is the real number such that*

$$\alpha(\mathcal{L}) = \frac{\lambda_2(\mathcal{L})}{\lambda_1(\mathcal{L})}.$$

Remark 5 (Unicity). To assure unicity of the solution and hence, removing the decryption failure, lattice-based cryptosystems generally use gap of at least $\alpha(\mathcal{L}) > 2$. Moreover, generally the gap of lattice used in cryptosystems are polynomial in its dimension.

Remark 6. As Hermite invariant is used to evaluate the resistance of a lattice, the bigger the gap of a lattice, the easier it will be practically to solve SVP or other problem on it. For a recent analysis of practical attacks against lattice with a big gap, please refer to [50].

As SVP is NP-hard, a relaxation factor has been introduced in the initial SVP to be able to propose and evaluate the quality of the polynomial algorithms.

Problem 2 (γ-SVP). Let $\mathcal{L}$ a lattice and $\gamma \geq 1$ a real positive number. Then, the γ-SVP is to find a vector $u \in \mathcal{L}$ such that

$$0 < \|u\| \leq \gamma\lambda_1.$$

In 1982 [51], Lenstra, Lenstra and Lovasz proposed a powerful algorithm which have a time complexity polynomial in the dimension d . It is known as the LLL algorithm and this algorithm returns a solution for $\gamma-$SVP for $\gamma = 2^{\frac{d-1}{2}}$ where $d = dim(\mathcal{L})$[3]. This property leads to break cryptosystems using lattice with gap bigger than $2^{\frac{d-1}{2}}$. However, in practice, LLL seems to be much more efficient [48]. In addition, a lot of improvements have been proposed on LLL to obtain a better approximation factor and/or a better time complexity. For the recent result on LLL, we refer the readers to [52,53].

A second category of lattice problems are based on different values that the successive minima.

Definition 8 (Minimum Distance). *Let $\mathcal{L}$ a lattice and u a vector. The Minimum Distance of u to $\mathcal{L}$, denoted as $dist(u, \mathcal{L})$ is the smallest real number such that there exists a vector $v \in \mathcal{L}$ with $\|u - v\| = dist(u, \mathcal{L})$. The problem to find such a vector v is known as the Closest Vector Problem (CVP).*

Theorem 6 (Emde Boas, [54]). *CVP is NP-Hard.*

As for SVP, CVP has a relaxed version as defined as follows.

Problem 3 (γ-CVP). Let $\mathcal{L}$ a lattice, w a vector and $\gamma \geq 1$ a real positive number. The γ-CVP is to find a vector $u \in \mathcal{L}$, $\|w - u\| \leq \gamma dist(u, \mathcal{L})$.

In 1986, Babai [55] proposed two polynomial methods to solve $\gamma-$CVP: the *nearest plane* and the *round-off* methods. Those algorithms solve γ-CVP within $\gamma = 2^{\frac{d}{2}}$ and $\gamma = 1+2d\left(\frac{9}{2}\right)^{\frac{d}{2}}$, respectively. Babai's algorithms use an LLL-reduced basis. Consequently all the variants of LLL, including BKZ utilization [56] proposed by Schnorr, are naturally the improvement of Babai's methods.

Moreover, there exists an heuristic way introduced by Kannan [57] to directly solve γ-CVP using algorithm made to solve γ-SVP: the *embedding method* . Instead of solving γ-CVP, we solve γ-SVP in a different lattice. Finding the closest vector of v in $\mathcal{L}(B)$ can be done by solving the shortest vector of $\mathcal{L}(B')$ with $B' = \begin{pmatrix} B & 0 \\ v & 1 \end{pmatrix}$. This method has been successfully used by Nguyen [29] for constructing his first attack against GGH cryptosystem and it seems practically the best way to attack a CVP-based cryptosystem.

[3] With $\delta = 0.75$ for LLL utilization parameter.

3 Intersecting Lattices

Each attack proposed in this paper is inspired by a new general simplification method of lattice problems.

Theorem 7. *Let $\mathcal{L}_1, \mathcal{L}_2$ two lattices and v a vector such that v is a shortest vector of both $\mathcal{L}_1$ and $\mathcal{L}_2$. Then, v is a shortest vector of the lattice $\mathcal{L}_1 \cap \mathcal{L}_2$,*

$$\gamma(\mathcal{L}_1 \cap \mathcal{L}_2) \leq \gamma(\mathcal{L}_1), \gamma(\mathcal{L}_2)$$

and

$$\alpha(\mathcal{L}_1 \cap \mathcal{L}_2) \geq \alpha(\mathcal{L}_1), \alpha(\mathcal{L}_2).$$

Proof.

We prove that v is the shortest vector of $\mathcal{L}_1 \cap \mathcal{L}_2$.

As $v \in \mathcal{L}_1, \mathcal{L}_2$, we have $v \in \mathcal{L}_1 \cap \mathcal{L}_2$. Suppose that there exists a non-zero vector $v' \in \mathcal{L}_1 \cap \mathcal{L}_2$ such that $0 < \|v'\| < \|v\|$. As $v' \in \mathcal{L}_1 \cap \mathcal{L}_2$, we have $v' \in \mathcal{L}_1$ with $0 < \|v'\| < \|v\|$, which is impossible as v is the shortest non-zero vector of $\mathcal{L}_1$. We have proved that for any non-zero vector $v' \in \mathcal{L}_1 \cap \mathcal{L}_2$, $\|v\| \leq \|v'\|$: v is the shortest vector of $\mathcal{L}_1 \cap \mathcal{L}_2$.

We prove that $\gamma(\mathcal{L}_1 \cap \mathcal{L}_2) \leq \gamma(\mathcal{L}_1)$.

Let's compare $\gamma(\mathcal{L}_1 \cap \mathcal{L}_2)$ with $\gamma(\mathcal{L}_1)$. We have proved that $\lambda_1(\mathcal{L}_1 \cap \mathcal{L}_2) = \|v\| = \lambda_1(\mathcal{L}_1)$. As $\mathcal{L}_1 \cap \mathcal{L}_2 \subseteq \mathcal{L}_1$, we have $\dim(\mathcal{L}_1 \cap \mathcal{L}_2) \leq \dim(\mathcal{L}_1)$ and $\det(\mathcal{L}_1 \cap \mathcal{L}_2) \geq \det(\mathcal{L}_1)$. We obtain

$$\gamma(\mathcal{L}_1 \cap \mathcal{L}_2) = \left(\frac{\lambda_1(\mathcal{L}_1 \cap \mathcal{L}_2)}{\det(\mathcal{L}_1 \cap \mathcal{L}_2)^{1/dim(\mathcal{L}_1 \cap \mathcal{L}_2)}}\right)^2 \leq \left(\frac{\lambda_1(\mathcal{L}_1)}{\det(\mathcal{L}_1)^{1/dim(\mathcal{L}_1)}}\right)^2 = \gamma(\mathcal{L}_1).$$

The same proof can be performed with $\mathcal{L}_2$, and consequently, we obtain $\gamma(\mathcal{L}_1 \cap \mathcal{L}_2) \leq \gamma(\mathcal{L}_1), \gamma(\mathcal{L}_2)$.

We prove that $\alpha(\mathcal{L}_1 \cap \mathcal{L}_2) \geq \alpha(\mathcal{L}_1)$.

Let's compare $\alpha(\mathcal{L}_1 \cap \mathcal{L}_2)$ with $\alpha(\mathcal{L}_1)$. We have proved that $\lambda_1(\mathcal{L}_1 \cap \mathcal{L}_2) = \|v\| = \lambda_1(\mathcal{L}_1)$. Suppose that we have two independent vectors $v_1, v_2 \in \mathcal{L}_1 \cap \mathcal{L}_2$ such that $max(\|v_1\|, \|v_2\|) = \lambda_2(\mathcal{L}_1 \cap \mathcal{L}_2)$. Then, since v_1, v_2 are also two independent vectors of $\mathcal{L}_1$, we obtain $\lambda_2(\mathcal{L}_1) \leq max(\|v_1\|, \|v_2\|)$. We have $\lambda_2(\mathcal{L}_1) \leq max(\|v_1\|, \|v_2\|) = \lambda_2(\mathcal{L}_1 \cap \mathcal{L}_2)$. Finally, we obtain

$$\alpha(\mathcal{L}_1 \cap \mathcal{L}_2) = \left(\frac{\lambda_2(\mathcal{L}_1 \cap \mathcal{L}_2)}{\lambda_1(\mathcal{L}_1 \cap \mathcal{L}_2)}\right) \geq \left(\frac{\lambda_2(\mathcal{L}_1)}{\lambda_1(\mathcal{L}_1)}\right) = \alpha(\mathcal{L}_1).$$

The same proof can be performed with $\mathcal{L}_2$, and consequently, we obtain $\alpha(\mathcal{L}_1 \cap \mathcal{L}_2) \geq \alpha(\mathcal{L}_1), \alpha(\mathcal{L}_2)$. □

Theorem 7 is crucial as it demonstrates that to solve the shortest vector problem on the intersection of lattices will be at least easier that in the initial lattice. We will see in Section 5 that practical problems become a lot easier. Nevertheless, the practical efficiency can not be shown in a general theorem as Theorem 7. This is simply because if $\mathcal{L}_1 = \mathcal{L}_2$, we obtain $\mathcal{L}_1 \cap \mathcal{L}_2 = \mathcal{L}_1 = \mathcal{L}_2$,

$$\gamma(\mathcal{L}_1 \cap \mathcal{L}_2) = \gamma(\mathcal{L}_1) = \gamma(\mathcal{L}_2)$$

and

$$\alpha(\mathcal{L}_1 \cap \mathcal{L}_2) = \alpha(\mathcal{L}_1) = \alpha(\mathcal{L}_2).$$

Remark 7. For cryptosystems based on CVP, we will use the embedding method to model as a SVP before applying Theorem 7.

4 Practical Broadcast Attacks

In this section, we adapt the general method (Theorem 7) to different lattice or knapsack cryptosystems. For convenience, we will always firstly recall the 'challenge' in the cryptosystem involved followed by our proposed attack. All of our attacks are heuristic.

4.1 A Broadcast Attack on GGH Type A

Problem 4 (GGH$_A$ Challenge). Let $B \in \mathbb{Z}^{n,n}$ a basis and $c \in \mathbb{Z}^n$ a vector such that there exist two vectors $r, m \in \mathbb{Z}^n$ with $c = rB + m$. Then, the GGH$_A$ challenge (B, c) is to find m.

Algorithm 1. Broadcast Attack on GGH$_A$ Challenges

```
Input  : $(B_i, c_i)$ $k$ GGH$_A$ challenges.
Output: $m \in \mathbb{Z}^n$.
begin
    Compute $B_i' = \begin{pmatrix} B_i & 0 \\ c_i & 1 \end{pmatrix}$.
    Compute $\mathcal{L} = \bigcap_{i=1}^k \mathcal{L}(B_i')$.
    Find $\begin{pmatrix} m & 1 \end{pmatrix}$ shortest vector of $\mathcal{L}$.
end
```

Example 1. The initial proposition in [27] is obviously concerned by this attack. However, we will refer to Micciancio cryptosystems [31] as a non-broken cryptosystem that will also be susceptible against this attack.

4.2 A Broadcast Attack on GGH Type B

Problem 5 (GGH$_B$ Challenge). Let $B \in \mathbb{Z}^{n,n}$ a basis and $c \in \mathbb{Z}^n$ a vector such that there exist two vectors $m, r \in \mathbb{Z}^n$ with $c = mB + r$. Then, the GGH$_B$ challenge (B, c) is to find m.

The idea here is a bit different. As we have $mB_1+r_1=c_1$ and $mB_2+r_2=c_2$, we construct a third challenge $mB_3+r_3=c_3$ with $B_3=B_1+B_2$ and $c_3=c_1+c_2$. Practically, the fact that $\|r\|$ grows will be less important than the growth of B.

Algorithm 2. Broadcast Attack on GGH$_B$ Challenges

```
Input  : $(B_i, c_i)$ $k$ GGH$_B$ challenges.
Output: $m \in \mathbb{Z}^n$.
begin
    Compute $B = \sum_{i=1}^{k} B_i$.
    Compute $c = \sum_{i=1}^{k} c_i$.
    Find the closest vector $v$ of $c$ in $\mathcal{L}(B)$.
    Compute $m = vB^{-1}$.
end
```

Algorithm 2 do not use Theorem 7 and cannot be proved to have a simpler problem as the $\lambda_1(\mathcal{L}(B_1+B_2))$ can be bigger than $\lambda_1(\mathcal{L}(B_1))$. However, we will see than practically $\lambda_1(\mathcal{L}(B_1+B_2))$ will be bigger. Practically, we will also use the embedding method for the third step of Algorithm 2.

Example 2. Cryptosystems concerned with this attack include [30] and the more recent work of [32].

4.3 A First Broadcast Attack on Knapsack Cryptosystems

Problem 6 (Knapsack Challenge). Let $a \in \mathbb{N}^n$ a positive integer vector and $s \in \mathbb{N}$ an integer such that there exists $m \in [0,1]^n$ a boolean vector such $ma^T = s$. Then, the Knapsack challenge (a, s) is to find m.

The attack proposed here is an adaptation of Algorithm 1 to the knapsack challenge as it has been already modelled by [10] in a lattice problem. Other modellings can been also adapted with the same technique.

Algorithm 3. Broadcast Attack on Knapsack Challenge

```
Input  : $(a_i, s_i)$ $k$ knapsack challenges.
Output: $m \in [0,1]^n$.
begin
    Compute $B_i = \begin{pmatrix} Id & a_i^T & 0 \\ 0 & s & 1 \end{pmatrix}$.
    Compute $\mathcal{L} = \bigcap_{i=1}^{k} \mathcal{L}(B_i)$.
    Find $\begin{pmatrix} m & 0 & 1 \end{pmatrix}$ shortest vector of $\mathcal{L}$.
end
```

Example 3. The examples of practical schemes that are susceptible against our attack are as follows. We refer to the survey of Odlysko [17] for knapsack cryptosystems that are susceptible against this attack. However, we also refer to [16] for one of the ‘rare’ non-broken knapsack cryptosystems that are also susceptible against this attack. The recent proposition of [58] is also concerned.

Remark 8. We remark that the dimension of $\mathcal{L}$ decreases further when k increases. Practically, we have $\dim(\mathcal{L}) = n - k$ with a high probability[4]. It is because each $\mathcal{L}(B_i)$ have a dimension smaller than n, $\dim(\mathcal{L}(B_i)) = n - 1$. This decrease will obviously stop at a dimension of 1 with a lattice of a basis only composed by $\begin{pmatrix} m \; 0 \; 1 \end{pmatrix}$.

4.4 A Second Broadcast Attack on Knapsack Cryptosystems

Inspired by the previous remark (Remark 8), we notice that if we have n equations $ma_i^T = s_i$, we can concatenate these equations to obtain $mA = s$ with $A \in \mathbb{Z}^{n,n}$ and $s \in \mathbb{Z}^n$. Moreover, these equations can be solved with high probability.

Algorithm 4. Broadcast Attack on Knapsack Challenges without Lattice Reduction

```
Input  : $(a_i, s_i)$ $n$ knapsack challenges.
Output: $m \in [0,1]^n$.
begin
    Compute $A = \begin{pmatrix} a_1^T \ldots a_n^T \end{pmatrix}$ .
    Compute $s = \begin{pmatrix} s_1 \ldots s_n \end{pmatrix}$ .
    Compute $m = sA^{-1}$.
end
```

The main advantage of our second attack is to avoid the use of any lattice reduction. The impact of this gain will enable us to use the attack in a huge dimension where the use of LLL will be computationally expensive. However, its drawback is the number of challenge required. The first attack will require practically less challenges to reveal the plaintext.

This method is also heuristic as A can be singular [5] and A^{-1} does not exist. However, probability of such a situation is extremely low and will be less probable with more knapsack challenges.

5 Practical Result

In this section, we present result of the previously presented techniques to attack different lattice-based cryptosystems. To perform these attacks, we use the embedding method with a lattice reduction done with LLL[6]. Cryptosystems and attacks were implemented under the MAGMA library [59]. Tests were made 20 times, for each 10 dimensions between 10 and 300. For each test, a random

[4] Under the probability that $\forall 1 \leq i_1, i_2 \leq k, 1 \leq j \leq n, a_{i_1}[j] \neq a_{i_2}[j]$.

[5] $det(A) = 0$.

[6] With $\delta = 0.9999$ for LLL utilization parameter.

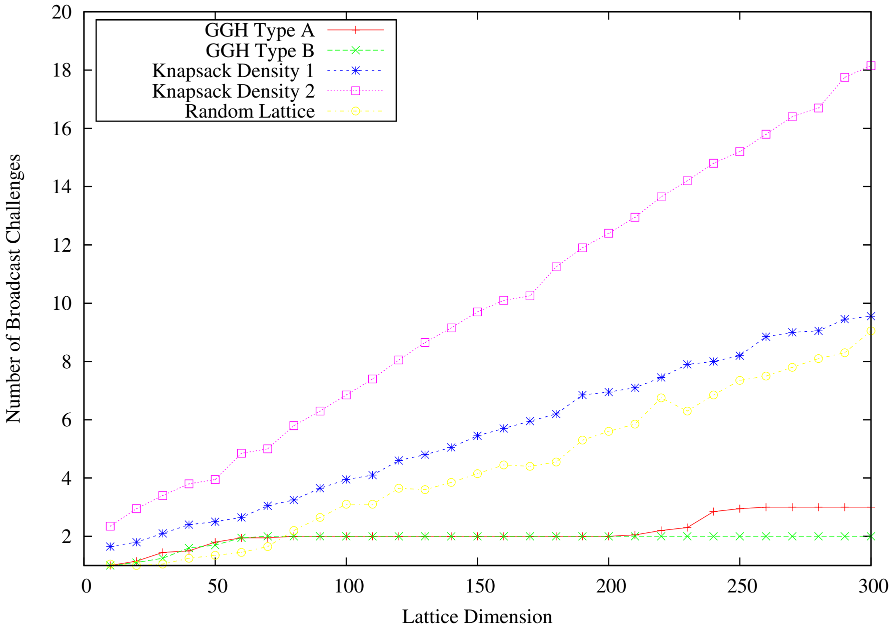


Fig. 1. Number of needed broadcast challenges to extract message from different cryptosystems

message is encrypted with a different random public basis repetitively until the attack is successful. Cryptosystems are implemented as close as possible to the initial paper. The list of different cryptosystems analyzed is as follows:

1. The initial GGH cryptosystem (Type A) attacked with Algorithm 1.
2. The second GGH cryptosystem (Type B) attacked with Algorithm 2.
3. A knapsack problem of density 1.0. This problem does not correspond to a real cryptosystem but to any knapsack cryptosystem which use such a problem. This attack is more general that the previous ones.
4. A knapsack problem of density 2.0. This problem is an extreme case. As we do not know if some trapdoor functions can be created for such a problem, for instance due to the reason of unicity. However, it gives us a security bound as problems with lower density will be easier to attack.
5. A random lattice CVP problem. This problem is the one which gives us a reference. For this one, we have created random lattice and a vector with $dist \sim \frac{\lambda_1}{2}$ to assure that at least the existence of a decryption algorithm. To create a random lattice, we use the same technique proposed in [48,50] using the results on random lattice from [47]. This problem corresponds to the general situation to a lattice-based cryptosystem which have the possibility to decrypt even if some trapdoors may not exist. This problem corresponds to a security upper bound for CVP based cryptosystems.

The purpose of those tests is not to give some security parameter bounds (as more powerfull SVP solver can be used than LLL, BKZ for example) but to show how evolves the difficulty of those problems with more and more challenges. Figure 1 summarizes our results.

6 Conclusion and Countermeasures

In this paper, we proposed an efficient way to simplify lattice problems which have the same solution. This technique leads to some heuristic and efficient attacks on the existing lattice or knapsack cryptosystems. However, some lattice-based cryptosystems naturally resist to those attacks. Ajtai-Dwork cryptosystem [19] and its different improvements, such as [21,22,23] or [24,25,26], are not concerned by our attacks. This is clearly due to the huge extension factor which allows those cryptosystems to put a strong part of random and hence, there is no common vector. For the same reason, the proposition of knapsack-based probabilistic encryption of [60] will be naturally resistant as well. In the same direction, we remark that after some tests, NTRU lattices should be extremely weak against intersecting lattices. However, the fact that half of its message is random leads to a complete protection against broadcast attacks. Those remarks inspired an obvious countermeasure. Concerned cryptosystems have just to add to their messages a random part (e.g. a hash of the public key itself) that is sufficiently big to prevent two messages to be equal under a reasonable probability. This is inline with the direction suggested in the traditional cryptography (e.g. [2,3]) to ensure the security in the IND-CCA sense. The cost of such counter measure is an expansion factor which have repercussion in both space and time complexity. If the solution was known before, the utility of such counter measure was never shown to be necessary. This is the result of this work. Nonetheless, if the solution is 'simple', some further techniques should be incorporated as for cryptosystems which resist to LLL attacks with two messages, after intersection even of only two messages, the new problem will be easier compared to the original problem.

Intersecting lattice has shown to be interesting to perform cryptanalysis. However, we believe that those kind of techniques can also lead to constructive utilization as original from other techniques used generally in cryptography.

References

1. Håstad, J.: Solving simultaneous modular equations of low degree. SIAM J. Comput. 17, 336–341 (1988)
2. Bellare, M., Boldyreva, A., Micali, S.: Public-key encryption in a multi-user setting: Security proofs and improvements. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 259–274. Springer, Heidelberg (2000)
3. Baudron, O., Pointcheval, D., Stern, J.: Extended notions of security for multicast public key cryptosystems. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 499–511. Springer, Heidelberg (2000)
4. Merkle, R.C., Hellman, M.E.: Hiding information and signatures in trapdoor knapsacks. IEEE Transactions on Information Theory IT-24, 525–530 (1978)

5. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory IT-22, 644–654 (1976)
6. Karp, K.M.: Reducibility among combinatorial problems. Complexity of Computer Computations (1972)
7. Shamir, A.: A polynomial time algorithm for breaking the basic merkle-hellman cryptosystem. In: CRYPTO, pp. 279–288 (1982)
8. Shamir, A.: A polynomial-time algorithm for breaking the basic merkle-hellman cryptosystem. IEEE Transactions on Information Theory 30, 699–704 (1984)
9. Adleman, L.M.: On breaking generalized knapsack public key cryptosystems (abstract). In: STOC, pp. 402–412 (1983)
10. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. Journal of the ACM 32, 229–246 (1985)
11. Coster, M.J., LaMacchia, B.A., Odlyzko, A.M.: An improved low-density subset sum algorithm. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 54–67. Springer, Heidelberg (1991)
12. Coster, M.J., Joux, A., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C.P., Stern, J.: Improved low-density subset sum algorithms. Computational Complexity 2, 111–128 (1992)
13. Schnorr, C.-P., Hörner, H.H.: Attacking the chor-rivest cryptosystem by improved lattice reduction. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 1–12. Springer, Heidelberg (1995)
14. Omura, K., Tanaka, K.: Density attack to the knapsack cryptosystems with enumerative source encoding. IEICE Trans. Fundam. Electron Commun. Comput. Sci. 87, 1564–1569 (2004)
15. Chor, B., Rivest, R.L.: A knapsack-type public key cryptosystem based on arithmetic in finite fields. IEEE Transactions on Information Theory 34, 901–909 (1988)
16. Okamoto, T., Tanaka, K., Uchiyama, S.: Quantum public-key cryptosystems. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 147–165. Springer, Heidelberg (2000)
17. Odlyzko, A.M.: The rise and fall of knapsack cryptosystems. Cryptology and Computational Number Theory 42, 75–88 (1990)
18. Nguyen, P.Q., Stern, J.: Adapting density attacks to low-weight knapsacks. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 41–58. Springer, Heidelberg (2005)
19. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: Twenty-Ninth Annual ACM Symposium on the Theory of Computing (STOC 1997), pp. 284–293 (1997)
20. Nguyen, P.Q., Stern, J.: Cryptanalysis of the ajtai-dwork cryptosystem. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 223–242. Springer, Heidelberg (1998)
21. Goldreich, O., Goldwasser, S., Halevi, S.: Eliminating decryption errors in the ajtai-dwork cryptosystem. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 105–111. Springer, Heidelberg (1997)
22. Cai, J.-Y., Cusick, T.W.: A lattice-based public-key cryptosystem. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 219–233. Springer, Heidelberg (1999)
23. Kawachi, A., Tanaka, K., Xagawa, K.: Multi-bit cryptosystems based on lattice problems. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 315–329. Springer, Heidelberg (2007)
24. Regev, O.: Improved inapproximability of lattice and coding problems with preprocessing. In: IEEE Conference on Computational Complexity, pp. 363–370 (2003)

25. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC, pp. 84–93 (2005)
26. Ajtai, M.: Representing hard lattices with o(n log n) bits. In: STOC, pp. 94–103 (2005)
27. Goldreich, O., Goldwasser, S., Halevi, S.: Public-key cryptosystems from lattice reductions problems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 112–131. Springer, Heidelberg (1997)
28. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. Deep Space Network Progress Report 44, 114–116 (1978)
29. Nguyen, P.Q.: Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from crypto 1997. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 288–304. Springer, Heidelberg (1999)
30. Fischlin, R., Seifert, J.P.: Tensor-based trapdoors for cvp and their application to public key cryptography. In: IMA Int. Conf., 244–257 (1999)
31. Micciancio, D.: Improving lattice based cryptosystems using the Hermite normal form. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 126–145. Springer, Heidelberg (2001)
32. Paeng, S.H., Jung, B.E., Ha, K.C.: A lattice based public key cryptosystem using polynomial representations. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 292–308. Springer, Heidelberg (2003)
33. Han, D., Kim, M.-H., Yeom, Y.: Cryptanalysis of the paeng-jung-ha cryptosystem from pkc 2003. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 107–117. Springer, Heidelberg (2007)
34. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A ring-based public key cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998)
35. Coppersmith, D., Shamir, A.: Lattice attacks on ntru. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 52–61. Springer, Heidelberg (1997)
36. Micciancio, D., Goldwasser, S.: Complexity of Lattice Problems, A Cryptographic Perspective. Kluwer Academic Publishers, Dordrecht (2002)
37. Minkowski, H.: Geometrie der Zahlen. B. G. Teubner, Leipzig (1896)
38. Cassels, J.W.S.: An Introduction to The Geometry of Numbers. Springer, Heidelberg (1959)
39. Lovász, L.: An Algorithmic Theory of Numbers, Graphs and Convexity. In: CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 50. SIAM Publications, Philadelphia (1986)
40. Conway, J.H., Sloane, N.J.A.: Sphere Packings, Lattices and Groups. Springer, Heidelberg (1988)
41. Cohen, H.: A course in computational algebraic number theory. Graduate Texts in Mathematics, vol. 138. Springer, Heidelberg (1993)
42. Kannan, R., Bachem, A.: Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. SIAM Journal of Computing 8, 499–507 (1979)
43. Micciancio, D., Warinschi, B.: A linear space algorithm for computing the Hermite normal form. In: International Symposium on Symbolic Algebraic Computation (ISSAC 2001), pp. 231–236 (2001)
44. Ajtai, M.: The shortest vector problem in l_2 is NP-hard for randomized reductions (extended abstract). In: Thirtieth Annual ACM Symposium on the Theory of Computing (STOC 1998), pp. 10–19 (1998)
45. Ajtai, M.: Generating random lattices according to the invariant distribution (2006)

46. Ajtai, M.: Random lattices and a conjectured 0 - 1 law about their polynomial time computable properties. In: FOCS, pp. 733–742 (2002)
47. Goldstein, D., Mayer, A.: On the equidistribution of Hecke points. Forum Mathematicum 15, 165–189 (2003)
48. Nguyen, P.Q., Stehlé, D.: LLL on the average. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 238–256. Springer, Heidelberg (2006)
49. Nguyen, P.Q., Stern, J.: The two faces of lattices in cryptology. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 146–180. Springer, Heidelberg (2001)
50. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer, Heidelberg (2008)
51. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen 261, 513–534 (1982)
52. Nguyen, P.Q., Stehlé, D.: Floating-point LLL revisited. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 215–233. Springer, Heidelberg (2005)
53. Schnorr, C.P.: Fast LLL-type lattice reduction. Information and Computation 204, 1–25 (2006)
54. Boas, P.V.E.: Another NP-complete problem and the complexity of computing short vectors in lattices. Technical Report 81-04, Mathematics Department, University of Amsterdam (1981)
55. Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica 6, 1–13 (1986)
56. Schnorr, C.P.: Block reduced lattice bases and successive minima. Combinatorics, Probability & Computing 3, 507–522 (1994)
57. Kannan, R.: Minkowski's convex body theorem and integer programming. Math. Oper. Res. 12, 415–440 (1987)
58. Murakami, Y., Nasako, T.: Knapsack public-key cryptosystem using chinese remainder theorem. IACR ePrint Archive (2007)
59. Bosma, W., Cannon, J., Playoust, C.: The magma algebra system. i. the user language. J. Symobolic Computation 24, 235–265 (1997)
60. Wang, B., Wu, Q., Hu, Y.: A knapsack-based probabilistic encryption scheme. Inf. Sci. 177, 3981–3994 (2007)

Partial Key Exposure Attack on CRT-RSA

Santanu Sarkar and Subhamoy Maitra

Indian Statistical Institute, 203 B T Road, Kolkata 700 108, India
{santanu_r,subho}@isical.ac.in

Abstract. Consider CRT-RSA with $N = pq$, $q < p < 2q$, public encryption exponent e and private decryption exponents d_p, d_q. Jochemsz and May (Crypto 2007) presented that CRT-RSA is weak when d_p, d_q are smaller than $N^{0.073}$. As a follow-up work of that paper, we study the partial key exposure attack on CRT-RSA when some Most Significant Bits (MSBs) of d_p, d_q are exposed. Further, better results are obtained when a few MSBs of p (or q) are available too. We present theoretical results as well as experimental evidences to justify our claim. We also analyze the case when the decryption exponents are of different bit sizes and it is shown that CRT-RSA is more insecure in this case (than the case of d_p, d_q having the same bit size) considering the total bit size of d_p, d_q.

Keywords: RSA, CRT-RSA, Cryptanalysis, Factorization, Lattice, LLL Algorithm, Side Channel Attacks, Weak Keys.

1 Introduction

RSA [20] is one of the most popular cryptosystems in the history of this subject. Let us first briefly describe the idea of RSA:

- primes p, q, (generally the primes are considered to be of same bit size, i.e., $q < p < 2q$);
- $N = pq$, $\phi(N) = (p - 1)(q - 1)$;
- e, d are such that $ed = 1 + k\phi(N)$, $k \geq 1$;
- N, e are publicly available and the plaintext $M \in \mathbb{Z}_N$ is encrypted as $C \equiv M^e \bmod N$;
- the secret key d is required to decrypt the ciphertext $C \in \mathbb{Z}_N$ as $M \equiv C^d \bmod N$.

The study of RSA is one of the most attractive areas in cryptology research as evident from many excellent works (one may refer [4,14,19] and the references therein for detailed information).

Speeding up RSA encryption and decryption is of serious interest and for large N, both e, d cannot be small at the same time. For fast encryption, it is possible to use smaller e and e as small as $2^{16} + 1$ is widely believed to be a good candidate. For fast decryption, the value of d needs to be small. However, Wiener [21] showed that when $d < \frac{1}{3}N^{\frac{1}{4}}$ then N can easily be factored. Later,

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 473–484, 2009.

Boneh-Durfee [5] increased this bound up to $d < N^{0.292}$. Thus the use of smaller d is in general not recommended. In this direction, an alternative approach has been proposed by Wiener [21] exploiting the Chinese Remainder Theorem (CRT) for decryption. The idea is as follows:

– the public exponent e and the private CRT exponents d_p and d_q are used satisfying $ed_p \equiv 1 \bmod (p-1)$ and $ed_q \equiv 1 \bmod (q-1)$;
– the encryption of the plaintext $M \in \mathbb{Z}_N$ is same as the standard RSA;
– to decrypt a ciphertext $C \in \mathbb{Z}_N$ one needs to compute $M_1 \equiv C^{d_p} \bmod p$ and $M_2 \equiv C^{d_q} \bmod q$;
– using CRT, one can get the plaintext M such that $M \equiv M_1 \bmod p$ and $M \equiv M_2 \bmod q$.

This variant of RSA is popularly known as CRT-RSA. Without loss of generality, consider d_p is available. One can take any random integer a in $[2, N-1]$ and then $\gcd(a^{ed_p} - a, N)$ provides p with a probability almost equal to 1 (but not exactly 1). Thus, it is clear that CRT-RSA becomes insecure if any of the decryption exponents is known. An important work in this direction shows that with the availability of decryption oracle under a fault model, one factorize N in poly$(\log N)$ time [6, Section 2.2] and the idea has been improved by A. Lenstra [6, Section 2.2, Reference 16].

May [18] described two weaknesses in CRT-RSA that work when the smaller prime factor is less than $N^{0.382}$. Bleichenbacher and May [1] improved the idea of [18] when the smaller prime factor is less than $N^{0.468}$. In [12], at attack on CRT-RSA has been presented for small e when the primes are of the same bit size. Recently, Jochemsz and May [16] presented an attack on CRT-RSA with primes of same bit size in poly$(\log N)$ time. In [16], it is shown that CRT-RSA can be attacked when the encryption exponents are of the order of N, and d_p and d_q are smaller than $N^{0.073}$. The strategy of [16] is based on the idea presented in [15] which in turn exploits the techniques from [9]. Further, in [15], it has been shown that CRT-RSA is weak if $d_p - d_q$ is known and d_p, d_q are smaller than $N^{0.099}$.

We work with techniques similar to [16], but our analysis considers that certain amounts of MSBs of d_p, d_q are exposed. This model is already accepted in literature for analysis of standard RSA, where it is considered that certain fraction of bits of the secret decryption exponent d may be exposed [3,2,11] by side channel attack. We consider a similar model in this paper. In addition, we also consider that a few MSBs of the secret prime p may be available, that can be exhaustively searched or may be known from side channel attack (as p, q are used during the decryption of CRT-RSA).

The main result of [16] was to show that for e of $O(N)$, CRT-RSA is insecure when d_p and d_q are smaller than $N^{0.073}$. Our generalization (see Theorem 2 and also Table 1 in Section 2) shows that if around $0.009 \log_2 N$ MSBs of each of d_p, d_q are exposed and $0.01 \log_2 N$ MSBs of p can be searched, then CRT-RSA is insecure when d_p and d_q are smaller than $N^{0.083}$. Our results are indeed not surprising, but the analysis we present in this paper give a clear indication how the results of [16] extend when certain amount of partial information is available regarding the secret parameters. Our theoretical ideas are supported by experimental evidences and the results are presented in Section 2.1. The case of unbalanced decryption exponents is considered in Section 3. Section 4 concludes the paper.

1.1 Preliminaries

Let us present some basics on lattice reduction techniques. Consider the linearly independent vectors $u_1, \ldots, u_\omega \in \mathbb{Z}^n$, where $\omega \leq n$. A lattice, spanned by $\{u_1, \ldots, u_\omega\}$, is the set of all linear combinations of $u_1, \ldots, u_\omega$, i.e., ω is the dimension of the lattice. A lattice is called full rank when $\omega = n$. Let L be a lattice spanned by the linearly independent vectors $u_1, \ldots, u_\omega$, where $u_1, \ldots, u_\omega \in \mathbb{Z}^n$. By $u_1^*, \ldots, u_\omega^*$, we denote the vectors obtained by applying the Gram-Schmidt process [7, Page 81] to the vectors $u_1, \ldots, u_\omega$.

The determinant of L is defined as $\det(L) = \prod_{i=1}^{\omega} ||u_i^*||$, where $||.||$ denotes the Euclidean norm on vectors. Given a polynomial $g(x, y) = \sum a_{i,j} x^i y^j$, we define the Euclidean norm as $\| \; g(x, y) \; \| = \sqrt{\sum_{i,j} a_{i,j}^2}$ and infinity norm as $\| \; g(x, y) \; \|_\infty = \max_{i,j} |a_{i,j}|$.

It is known that given a basis $u_1, \ldots, u_\omega$ of a lattice L, the LLL algorithm [17] can find a new basis $b_1, \ldots, b_\omega$ of L with the following properties.

1. $\| \; b_i^* \; \|^2 \leq 2 \; \| \; b_{i+1}^* \; \|^2$, for $1 \leq i < \omega$.
2. For all i, if $b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$ then $|\mu_{i,j}| \leq \frac{1}{2}$ for all j.
3. $\| \; b_i \; \| \leq 2^{\frac{\omega(\omega-1)+(i-1)(i-2)}{4(\omega-i+1)}} \det(L)^{\frac{1}{\omega-i+1}}$ for $i = 1, \ldots, \omega$.

By $b_1^*, \ldots, b_\omega^*$, we mean the vectors obtained by applying the Gram-Schmidt process to the vectors $b_1, \ldots, b_\omega$.

In [8], techniques have been discussed to find small integer roots of polynomials in a single variable $\bmod\ n$, and of polynomials in two variables over the integers. The idea of [8] extends to more than two variables also, but the method becomes probabilistic. The following theorem is also relevant to the idea of [8].

Theorem 1. *[13] Let $g(x, y, z, v)$ be a polynomial which is a sum of ω many monomials. Suppose $g(x_0, y_0, z_0, v_0) \equiv 0 \bmod n$, where $|x_0| < X$, $|y_0| < Y$, $|z_0| < Z$ and $|v_0| < V$. If $\| \; g(xX, yY, zZ, vV) \; \| < \frac{n}{\sqrt{\omega}}$, then $g(x_0, y_0, z_0, v_0) = 0$ holds over integers.*

Considering the property 3 mentioned above with $i = 4$ and Theorem 1, the condition $2^{\frac{\omega^2-\omega+6}{4(\omega-3)}} \det(L)^{\frac{1}{\omega-3}} < \frac{n}{\sqrt{\omega}}$ implies that if the polynomials b_1, b_2, b_3, b_4 (corresponding to the four shortest reduced basis vectors) have roots over 0 mod n, then those roots hold over integers too. The solutions corresponding to each unknown can be achieved by calculating the Gröbner basis of the ideal generated by $\{b_1, b_2, b_3, b_4\}$.

Suppose we have a set of polynomials $\{f_1, f_2, \ldots, f_i\}$ on n variables having the roots of the form $(x_{1,0}, x_{2,0}, \ldots, x_{n,0})$. Then it is known that the Gröbner

Basis [10, Page 77] $\{g_1, g_2, \ldots, g_j\}$, of $J = < f_1, f_2, \ldots, f_i >$ (the ideal generated by $\{f_1, f_2, \ldots, f_i\}$), preserves the set of common roots of $\{f_1, f_2, \ldots, f_i\}$. For our problems, we assume that the roots can be collected efficiently from $\{g_1, g_2, \ldots, g_j\}$. Though this is true in practice as noted from the experiments we perform, theoretically this may not always happen. Thus we formally state the following assumption that we will consider for the theoretical results.

Assumption 1. Consider a set of polynomials $\{f_1, f_2, \ldots, f_i\}$ on n variables having the roots of the form $(x_{1,0}, x_{2,0}, \ldots, x_{n,0})$. Let J be the ideal generated by $\{f_1, f_2, \ldots, f_i\}$. Then we will be able to collect the roots efficiently from the Gröbner Basis of J.

2 Weaknesses of CRT-RSA When Some MSBs of d_p, d_q and p Are Known

In this section, we extend the idea of [16] towards a partial key exposure attack on CRT-RSA where the secret primes are of the same bit size. We present a general result considering that some of the MSBs of d_p, d_q, p will be exposed.

Since $ed_p \equiv 1 \bmod (p-1)$ and $ed_q \equiv 1 \bmod (q-1)$, we write $ed_p = 1 + k(p-1)$ and $ed_q = 1 + l(q-1)$. We start with the following technical result.

Lemma 1. *Let $e = N^\alpha$ and $d_p, d_q < N^\delta$. Consider that d_{p_0}, d_{q_0}, p_0 are exposed such that $|d_p - d_{p_0}| < N^\gamma$, $|d_q - d_{q_0}| < N^\gamma$ and $|p - p_0| < N^\beta$. Then one can find the integers k_0, l_0 such that $|k - k_0|$ and $|l - l_0|$ are $O(N^\lambda)$ where $\lambda = \max\{\alpha + \delta + \beta - 1, \alpha + \gamma - \frac{1}{2}\}$.*

Proof. We consider p, q are of same bit size, i.e., $q < p < 2q$. In such a case, $\sqrt{N} < p < \sqrt{2N}$ and $\sqrt{\frac{N}{2}} < q < \sqrt{N}$. Estimate k_0 as the closest integer value of $\frac{ed_{p_0}-1}{p_0-1}$. Also we have $k = \frac{ed_p-1}{p-1}$. Now

$$\begin{aligned}
|k - k_0| &\approx |\frac{ed_p - 1}{p-1} - \frac{ed_{p_0} - 1}{p_0 - 1}| \\
&\approx |\frac{ed_p}{p} - \frac{ed_{p_0}}{p_0}| \\
&= |\frac{ed_p p_0 - ed_p p + ed_p p - ed_{p_0} p}{pp_0}| \\
&\leq \frac{ed_p|p - p_0| + ep|d_p - d_{p_0}|}{pp_0} \\
&< \frac{N^{\alpha+\delta+\beta} + \sqrt{2}N^{\alpha+\frac{1}{2}+\gamma}}{pp_0} \text{ (as } p < \sqrt{2N}\text{)} \\
&< N^{\alpha+\delta+\beta-1} + \sqrt{2}N^{\alpha+\gamma-\frac{1}{2}} \text{ (as } pp_0 > N\text{)} \\
&< (1 + \sqrt{2})N^\lambda,
\end{aligned}$$

where $\lambda = \max\{\alpha + \delta + \beta - 1, \alpha + \gamma - \frac{1}{2}\}$. Next we calculate $q_0 = \frac{N}{p_0}$. One can check $|q - q_0| < N^\beta$. Taking l_0 as the nearest integer of $\frac{ed_{q_0}-1}{q_0-1}$, it can be shown similarly as before that $|l - l_0|$ is $O(N^\lambda)$. □

Now we will prove our main result.

Theorem 2. *Let $e = N^\alpha$ and $d_p, d_q < N^\delta$. Consider that d_{p_0}, d_{q_0}, p_0 are exposed such that $|d_p - d_{p_0}| < N^\gamma$, $|d_q - d_{q_0}| < N^\gamma$ and $|p - p_0| < N^\beta$. Let $\lambda = \max\{\alpha + \delta + \beta - 1, \alpha + \gamma - \frac{1}{2}\}$. Then, under Assumption 1, one can factor N in poly$(\log N)$ time when*

$$\gamma < \max_{\tau \geq 0} h(\tau), \text{ where } h(\tau) = \frac{(2\alpha - 3\lambda)\tau^2 + (2\alpha - \frac{10}{3}\lambda)\tau + (\frac{\alpha}{2} - \frac{5}{6}\lambda)}{2\tau^3 + \frac{5}{2}\tau^2 + \frac{4}{3}\tau + \frac{1}{3}}.$$

Proof. Suppose d_{p_0}, d_{q_0}, p_0 are exposed from d_p, d_q and p respectively. Following Lemma 1, we get the approximations k_0, l_0 of k, l respectively. Let $d_{p_1} = d_p - d_{p_0}$, $d_{q_1} = d_q - d_{q_0}$, $k_1 = k - k_0$ and $l_1 = l - l_0$. Thus, $d_{p_1}, d_{q_1}, k_1, l_1$ are unknown to the attacker.

We have $ed_p = 1 + k(p-1)$ and $ed_q = 1 + l(q-1)$. This can be re-written as $ed_p + k - 1 = kp$ and $ed_q + l - 1 = lq$. Multiplying these two equations, we get

$$e^2 d_p d_q + ed_p(l-1) + ed_q(k-1) - (N-1)kl - (k+l-1) = 0.$$

Now putting $d_p = d_{p_1} + d_{p_0}$, $d_q = d_{q_1} + d_{q_0}$, $k = k_0 + k_1$ and $l = l_0 + l_1$ in the above equation we have $e^2 d_{p_1} d_{q_1} + (e^2 d_{p_0} - e + ek_0) d_{q_1} + (e^2 d_{q_0} - e + el_0) d_{p_1} + ek_1 d_{q_1} + el_1 d_{p_1} + (ed_{q_0} - 1 - l_0 N + l_0) k_1 + (ed_{p_0} - 1 - k_0 N + k_0) l_1 + (1 - N) k_1 l_1 + R = 0$, where $R = (e^2 d_{p_0} d_{q}0 - ed_{p_0} - ed_{q_0} + 1 + ed_{p_0} l_0 + ed_{q_0} k_0 - l_0 k_0 N + l_0 k_0 - l_0 - k_0)$ is a known constant. Now if we substitute $d_{p_1}, d_{q_1}, k_1, l_1$ by x, y, z, v respectively then we have $e^2 xy + (e^2 d_{p_0} - e + ek_0) y + (e^2 d_{q_0} - e + el_0) x + ezy + evx + (ed_{q_0} - 1 - l_0 N + l_0) z + (ed_{p_0} - 1 - k_0 N + k_0) v + (1 - N) zv + R = 0$. Hence we have to find the solution $d_{p_1}, d_{q_1}, k_1, l_1$ of the polynomial $f(x, y, z, v) = e^2 xy + (e^2 d_{p_0} - e + ek_0) y + (e^2 d_{q_0} - e + el_0) x + ezy + evx + (ed_{q_0} - 1 - l_0 N + l_0) z + (ed_{p_0} - 1 - k_0 N + k_0) v + (1 - N) zv + R$. Note that this polynomial has the same monomials as of $f(x_1, x_2, x_3, x_4)$ presented in [16, Section 4], though the coefficients are different. Also, the upper bounds on the variables are different as mentioned below.

Here $d_{p_1} < N^\gamma, d_{q_1} < N^\gamma$. Also from Lemma 1, k_1, l_1 are $O(N^\lambda)$. Let $X = Y = N^\gamma$, and $Z = V = N^\lambda$, which are the upper bounds of x, y, z, v respectively (note that for the upper bounds of z, v, we have neglected the constant terms as mentioned above).

When e is significantly greater than $N^{0.5}$, then d_{p_1}, d_{q_1} are significantly smaller than k_1, l_1. As we are mostly interested for large e, we apply extra shifts on x, y as advised in the "Extended Strategy" of [15, Page 274]. In this direction we define the following as in [16]:

$$S = \bigcup_{0 \leq j \leq t} \{x^{i_1+j} y^{i_2+j} z^{i_3} w^{i_4} : x^{i_1} y^{i_2} z^{i_3} w^{i_4} \text{ is a monomial of } f^{m-1}\},$$

$$M = \{\text{monomials of } x^{i_1} y^{i_2} z^{i_3} w^{i_4} f : x^{i_1} y^{i_2} z^{i_3} w^{i_4} \in S\}.$$

That is, $x^{i_1}y^{i_2}z^{i_3}v^{i_4} \in S$ iff $i_1 = 0, \ldots, m-1-i_3+t, i_2 = 0, \ldots, m-1-i_4+t, i_3 = 0, \ldots, m-1, i_4 = 0, \ldots, m-1$, and

$x^{i_1}y^{i_2}z^{i_3}v^{i_4} \in M$ iff $i_1 = 0, \ldots, m-i_3+t, i_2 = 0, \ldots, m-i_4+t, i_3 = 0, \ldots, m, i_4 = 0, \ldots, m$ for some non-negative integer t.

We need to find at least three more polynomials f_0, f_1, f_2 that share the same root $(d_{p_1}, d_{q_1}, k_1, l_1)$ over the integers. Given $W = ||f(xX, yY, zZ, vV)||_\infty$, from [15], we know that these polynomials can be found by lattice reduction if $X^{s_1}Y^{s_2}Z^{s_3}V^{s_4} < W^s$ for $s_r = \sum_{x^{i_1}y^{i_2}z^{i_3}v^{i_4} \in M \setminus S} i_r$, $r = 1, 2, 3, 4$ and $s = |S|$.

For a given integer m and $t = \tau m$, from the definition of S and M and neglecting the lower order terms we have the required condition same as the one presented in [16, Section 4] due to the same polynomial f used in both the cases.

$$(XY)^{\frac{5}{12}+\frac{5}{3}\tau+\frac{9}{4}\tau^2+\tau^3}(ZV)^{\frac{5}{12}+\frac{5}{3}\tau+\frac{3}{2}\tau^2} < W^{\frac{1}{4}+\tau+\tau^2}. \tag{1}$$

However, the bounds on X, Y, Z, V, W are different than what presented in [16, Section 4]. As $W \geq N^{2\alpha+2\gamma}$, substituting the values of X, Y, Z, V in Inequality (1), it is enough to satisfy the following inequality:

$$(\frac{5}{12}+\frac{5}{3}\tau+\frac{9}{4}\tau^2+\tau^3)2\gamma + (\frac{5}{12}+\frac{5}{3}\tau+\frac{3}{2}\tau^2)2\lambda < (\frac{1}{4}+\tau+\tau^2)\cdot(2\alpha+2\gamma). \tag{2}$$

Thus we get the following:

$$\gamma < \frac{(2\alpha-3\lambda)\tau^2+(2\alpha-\frac{10}{3}\lambda)\tau+(\frac{\alpha}{2}-\frac{5}{6}\lambda)}{2\tau^3+\frac{5}{2}\tau^2+\frac{4}{3}\tau+\frac{1}{3}}.$$

Fixing α, λ, let $h(\tau) = \frac{(2\alpha-3\lambda)\tau^2+(2\alpha-\frac{10}{3}\lambda)\tau+(\frac{\alpha}{2}-\frac{5}{6}\lambda)}{2\tau^3+\frac{5}{2}\tau^2+\frac{4}{3}\tau+\frac{1}{3}}$. Putting $h'(\tau) = 0$, we get the equation

$$(6\lambda-4\alpha)\tau^4+(\frac{40\lambda}{3}-8\alpha)\tau^3+(\frac{28\lambda}{3}-\frac{16\alpha}{3})\tau^2+(\frac{13\lambda}{6}-\frac{7\alpha}{6})\tau = 0. \tag{3}$$

The non-negative real solutions of τ from this equation are considered and let τ_m be the value among them for which $h(\tau)$ is maximum. Putting this optimal value of τ we have $\gamma < \frac{(2\alpha-3\lambda)\tau_m{}^2+(2\alpha-\frac{10}{3}\lambda)\tau_m+(\frac{\alpha}{2}-\frac{5}{6}\lambda)}{2\tau_m{}^3+\frac{5}{2}\tau_m{}^2+\frac{4}{3}\tau_m+\frac{1}{3}}$.

Under Assumption 1, we get the root using Gröbner Basis as it is done in [16] and the algorithm works in poly$(\log N)$ time. □

It can be checked that when $\beta = \frac{1}{2}$ and $\gamma = \delta$, we have the same bound as in [16].

Below we present some numerical results based on Theorem 2. We start from $\alpha = 0.4$ as the results of [16] are better than the results of [12] when $\alpha \geq 0.4$ and we follow the the technique of [16] only. Additionally we like to mention that in the proof of Theorem 2, we have assumed that e is significantly greater than $N^{0.5}$, and that actually motivates the extra shifts on the variables x, y. Thus for $e < N^{0.5}$, the results may not be optimal. While studying the numerical results, we explain two cases:

Table 1. Increased bounds of decryption exponents (that are not secure) with knowledge of some MSBs of d_p, d_q with(out) the knowledge of some MSBs of p

α	δ following		$\delta - \gamma$, when		τ_m from Theorem 2	
	[16, Section 4.1]	Theorem 2	$\beta = \frac{1}{2}$	$\beta = \frac{1}{2} - 0.01$	$\beta = \frac{1}{2}$	$\beta = \frac{1}{2} - 0.01$
0.4	0.243	0.253	0.036	0.011	0	0
0.5	0.214	0.224	0.034	0.01	0	0
0.577	0.192	0.202	0.034	0.01	0	0
0.7	0.157	0.167	0.035	0.01	0	0
0.8	0.128	0.138	0.033	0.01	0.0708	0
0.9	0.1	0.11	0.032	0.01	0.2814	0.1479
0.925	0.093	0.103	0.031	0.01	0.3411	0.1972
0.95	0.087	0.097	0.032	0.012	0.4212	0.2626
1.0	0.073	0.083	0.027	0.01	0.5563	0.3751

1. when some of the MSBs of d_p, d_q are known, but none of the bits of p is known,
2. when some of the MSBs of d_p, d_q as well as p are known.

Let us now present Table 1 based on the numerical values arising out of Theorem 2 and compare it with the values presented in [16]. We consider the asymptotic upper bound of δ presented in the table in [16, Section 7.1] (it follows the formula of [16, Section 4.1]). As we claim to improve the bound on δ with knowledge of some MSBs of d_p, d_q, we take the δ values 0.01 more than the asymptotic upper bounds presented in [16].

To get the improved bounds on δ, we need to know $(\delta - \gamma) \log_2 N$ MSBs for each of the decryption exponents. We present the values of τ_m (as in the proof of Theorem 2) given different values of α, where $h'(\tau_m) = 0$ and $h(\tau_m)$ is maximum.

The exercises are done in both the cases, (i) when none of the MSBs of p is known, i.e., $\beta = \frac{1}{2}$ and (ii) when certain amount of MSBs ($0.01 \log_2 N$ bits) of p is known.

2.1 Experimental Results

We have implemented the programs in SAGE 3.1.1 over Linux Ubuntu 8.04 on a laptop with Dual CORE Intel(R) Pentium(R) D CPU 1.83 GHz, 2 GB RAM and 2 MB Cache.

As we work with low lattice dimensions, the theoretical bounds of d_p, d_q presented in Theorem 2 may not be reached and the actual requirement of MSBs to be known will be higher in experimental results than the numerical values arrived from the theoretical results. However, we show that the values of d_p, d_q achieved in our experimental results indeed exceed the experimental evidences presented in [16]. The implementation in [16, Section 5] used Coppersmith's original method [8] instead of Coron's reformulation [9]. On the other hand, we have followed the idea of [15] based on Coron's strategy [9] itself for our experiments.

Example 1. We consider 500 bits p, q, i.e., 1000 bits $N = pq$. The primes p, q are as follows:
257982890708293518390079668089547327140094192761354318342177532056
591406252118442411708524558220040883218254274670592143045254411833
7323748167716006673,
219175753591795206167656820069977488073370238295740745766814008472
138727195704836927992992835177976408563857693206444747859754365066
1348649341629849809.

We consider $\alpha = 0.8$, i.e., e is an 800 bit integer as follows:
381488272445274098681501407517305078463396289118603070856927451668
680476133479793702456098517534653338608646458073266532528791321957
770232953077386115965246285769851351453842444699396326285020512454
7833724025378461669383099996626917921859531.

In [16, Section 7.2], it has been shown that in such a case, decryption exponents up to 79 bits are insecure in practice. The lattice parameters used in this case are $m = 2, t = 0$ and the time required to run the LLL algorithm was 2 seconds in the experimental set up of [16].

The experiment is with decryption exponents of 90 bits, where d_p, d_q are 1187824505872763330365347843, 1197585192151765825516761747 respectively. We consider that 36 MSBs of each of d_p, d_q and 10 MSBs of p are known. We used the lattice parameters $m = 2, t = 0$. The time required to run the LLL algorithm is 24 seconds in our experimental set up.

As referred in the proof of Theorem 2, we have f, f_0, f_1, f_2 after the LLL algorithm. Then we apply the strategy exploiting Gröbner Basis and find a polynomial on the single variable v, i.e., l_1. Once we get l_1, we find l. Consequently we can find out q since $e > N^{\frac{1}{4}}$ (one may refer to the discussion in [16, Section 7.1]). □

Example 2. We consider 500 bits p, q, i.e., 1000 bits $N = pq$. The primes p, q are as follows:
308536652523786752403262271587380862779156002539534598580650377819
837308083923866689819772202260205864471663039568900406887433048818
1354605267758401737,
174884640339050989948134239058179947355258107199237642863944051459
787872529871998850799955677509191511939485741856556131422997911453
4175281806597704419.

We consider $\alpha = 1$, i.e., e is a 1000 bit integer as follows:
650836990581869614252071497849163666992784539277711863628883327803
447764262103970654043363725365856415076337791404005870468638937810
821150247754816434038612733008679525086364394895223068051664347774
104165881644840283525895285922376024682931696898353489945338018493
5253711023618697785834142726686557409.

In [16, Section 7.2], it has been shown that in such a case, decryption exponents up to 15 bits are insecure in practice. The lattice parameters used in this case are $m = 3, t = 1$ and the time required to run the LLL algorithm was 13787 seconds in the experimental set up of [16].

We consider the decryption exponents of 18 bits, where d_p, d_q are 255025, 257539 respectively. We consider that 9 MSBs of each of d_p, d_q, p are known. We used the lattice parameters $m = 2, t = 1$. The time required to run the LLL algorithm is 3347 seconds in our experimental set up. □

The values of d_p, d_q in Example 2 are quite low and any one of them can be easily searched for a complete attack. Example 2 is presented only for the purpose of comparison with the result of [16].

3 Unbalanced Decryption Exponents

In this section we present similar analysis as in the earlier section, with the only difference that now the decryption exponents d_p, d_q can be of different size. Instead of considering t amount of extra shifts on both the variables x, y as in Theorem 2, here we apply two different shifts t_1, t_2 on x, y respectively. Taking two different shifts produce better results than considering the same shift in case of unbalanced decryption exponents.

Theorem 3. *Let $e = N^{\alpha}$, $d_p < N^{\delta_1}$ and $d_q < N^{\delta_2}$. Consider that d_{p_0}, d_{q_0}, p_0 are exposed such that $|d_p - d_{p_0}| < N^{\gamma_1}$, $|d_q - d_{q_0}| < N^{\gamma_2}$ and $|p - p_0| < N^{\beta}$. Let $\lambda_1 = \max\{\alpha + \delta_1 + \beta - 1, \alpha + \gamma_1 - \frac{1}{2}\}$ and $\lambda_2 = \max\{\alpha + \delta_2 + \beta - 1, \alpha + \gamma_2 - \frac{1}{2}\}$. Then, under Assumption 1, one can factor N in poly$(\log N)$ time when there exist non-negative real numbers $\tau_1, \tau_2 \geq 0$ for which $h(\tau_1, \tau_2, \gamma_1, \gamma_2, \lambda_1, \lambda_2, \alpha) = {\tau_1}^2\tau_2\gamma_1 + \tau_1{\tau_2}^2\gamma_2 + \frac{3}{4}{\tau_1}^2\gamma_1 + \frac{1}{2}\tau_1\tau_2(\gamma_1 + \gamma_2) + \frac{3}{4}{\tau_2}^2\gamma_2 + \frac{3}{2}\tau_1\tau_2(\lambda_1 + \lambda_2) - 2\tau_1\tau_2\alpha + \frac{1}{2}\tau_1\gamma_1 + \frac{1}{6}\tau_2\gamma_1 + \frac{1}{6}\tau_1\gamma_2 + \frac{1}{2}\tau_2\gamma_2 + \tau_1\lambda_1 + \frac{2}{3}\tau_2\lambda_1 + \frac{2}{3}\tau_1\lambda_2 + \tau_2\lambda_2 - \tau_1\alpha - \tau_2\alpha + \frac{1}{6}(\gamma_1 + \gamma_2) + \frac{5}{12}(\lambda_1 + \lambda_2) - \frac{\alpha}{2} < 0$.*

Proof. This proof is similar to the proof of Theorem 2 till the construction of the polynomial $f(x, y, z, v)$.

Here $d_{p_1} < N^{\gamma_1}, d_{q_1} < N^{\gamma_2}$ and we also consider $k_1 < N^{\lambda_1}, l_1 < N^{\lambda_2}$ (ignoring the constants presented in Lemma 1). Let $X = N^{\gamma_1}, Y = N^{\gamma_2}, Z = N^{\lambda_1}, V = N^{\lambda_2}$.

We have the following definitions of S, M, where t_1, t_2 are non-negative integers.

$$S = \bigcup_{0 \leq j_1 \leq t_1, 0 \leq j_2 \leq t_2} \{x^{i_1+j_1}y^{i_2+j_2}z^{i_3}w^{i_4} : x^{i_1}y^{i_2}z^{i_3}w^{i_4} \text{ is a monomial of } f^{m-1}\},$$

$$M = \{\text{monomials of } x^{i_1}y^{i_2}z^{i_3}w^{i_4}f : x^{i_1}y^{i_2}z^{i_3}w^{i_4} \in S\}.$$

Similar to the proof of Theorem 2, we need, $X^{s_1}Y^{s_2}Z^{s_3}V^{s_4} < W^s$ for $s_r = \sum_{x^{i_1}y^{i_2}z^{i_3}v^{i_4} \in M \setminus S} i_r$, $r = 1, 2, 3, 4$, $s = |S|$ and $W = ||f(xX, yY, zZ, vV)||_{\infty} \geq N^{2\alpha + \gamma_1 + \gamma_2}$.

For a given integer m, let $t_1 = \tau_1 m$ and $t_2 = \tau_2 m$. Then from the definitions of S, M we have the required condition

$$X^{s_1}Y^{s_2}Z^{s_3}V^{s_4} < W^s, \tag{4}$$

where,
$s_1 = (\frac{5}{12}m^4 + m^3t_1 + \frac{3}{4}m^2{t_1}^2 + \frac{2}{3}m^3t_2 + \frac{3}{2}m^2t_1t_2 + m{t_1}^2t_2) + o(m^4)$,
$s_2 = (\frac{5}{12}m^4 + m^3t_2 + \frac{3}{4}m^2{t_2}^2 + \frac{2}{3}m^3t_1 + \frac{3}{2}m^2t_1t_2 + mt_1{t_2}^2) + o(m^4)$,
$s_3 = \frac{5}{12}m^4 + m^3t_1 + \frac{2}{3}m^3t_2 + \frac{3}{2}m^2t_1t_2 + o(m^4)$,
$s_4 = \frac{5}{12}m^4 + m^3t_2 + \frac{2}{3}m^3t_1 + \frac{3}{2}m^2t_1t_2 + o(m^4)$, and
$s = \frac{m^4}{4} + \frac{m^3(t_1+t_2)}{2} + m^2t_1t_2 + o(m^4)$.

Substituting the values of X, Y, Z, V, W in Inequality (4), and putting $t_1 = \tau_1 m$, $t_2 = \tau_2 m$, we have $(\frac{5}{12} + \tau_1 + \frac{3}{4}{\tau_1}^2 + \frac{2}{3}\tau_2 + \frac{3}{2}\tau_1\tau_2 + {\tau_1}^2\tau_2)\gamma_1 + (\frac{5}{12} + \tau_2 + \frac{3}{4}{\tau_2}^2 + \frac{2}{3}\tau_1 + \frac{3}{2}\tau_1\tau_2 + \tau_1{\tau_2}^2)\gamma_2 + (\frac{5}{12} + \tau_1 + \frac{2}{3}\tau_2 + \frac{3}{2}\tau_1\tau_2)\lambda_1 + (\frac{5}{12} + \tau_2 + \frac{2}{3}\tau_1 + \frac{3}{2}\tau_1\tau_2)\lambda_2 < (\frac{1}{4} + \frac{\tau_1+\tau_2}{2} + \tau_1\tau_2)(2\alpha + \gamma_1 + \gamma_2)$. From which we get $h(\tau_1, \tau_2, \gamma_1, \gamma_2, \lambda_1, \lambda_2, \alpha) = {\tau_1}^2\tau_2\gamma_1 + \tau_1{\tau_2}^2\gamma_2 + \frac{3}{4}{\tau_1}^2\gamma_1 + \frac{1}{2}\tau_1\tau_2(\gamma_1 + \gamma_2) + \frac{3}{4}{\tau_2}^2\gamma_2 + \frac{3}{2}\tau_1\tau_2(\lambda_1 + \lambda_2) - 2\tau_1\tau_2\alpha + \frac{1}{2}\tau_1\gamma_1 + \frac{1}{6}\tau_2\gamma_1 + \frac{1}{6}\tau_1\gamma_2 + \frac{1}{2}\tau_2\gamma_2 + \tau_1\lambda_1 + \frac{2}{3}\tau_2\lambda_1 + \frac{2}{3}\tau_1\lambda_2 + \tau_2\lambda_2 - \tau_1\alpha - \tau_2\alpha + \frac{1}{6}(\gamma_1 + \gamma_2) + \frac{5}{12}(\lambda_1 + \lambda_2) - \frac{\alpha}{2} < 0$.

Then the proof follows by finding the root similar to the idea described in Theorem 2. □

One may check that putting $\tau_1 = \tau_2 = \tau$ in Theorem 3, we get the same form as presented in Theorem 2.

First consider the case, when no information about the bits of d_p, d_q, p is known. Thus, we have $\gamma_1 = \delta_1$, $\gamma_2 = \delta_2$, $\beta = \frac{1}{2}$. When δ_1, δ_2 are available, we will take the partial derivative of h with respect to τ_1, τ_2 and equate each of them to 0 to get non-negative solutions of τ_1, τ_2. Given any pair of such non-negative solutions, if h is less than zero, then for that δ_1, δ_2, CRT-RSA will be insecure.

Let us assume that for balanced d_p, d_q, CRT-RSA is insecure when $d_p, d_q < N^{\delta}$. On the other hand, consider that CRT-RSA is insecure for the unbalanced case when $d_p < N^{\delta_1}$, $d_p < N^{\delta_2}$. This situation is worth investigating when $2\delta < \delta_1 + \delta_2$. We find that this indeed happens. In [16], it has been shown that when e is $O(N)$, then CRT-RSA is insecure when $\delta = 0.073$. In Table 2, we find the cases when $\delta_1 + \delta_2$ is greater than $2\delta = 0.146$.

Table 2. Values for which CRT-RSA with unbalanced decryption exponents is insecure

δ_1	0.06	0.05	0.04	0.03
δ_2	0.087	0.099	0.111	0.126
$\delta_1 + \delta_2$	0.147	0.149	0.151	0.156

Thus considering the total amount of bits in the decryption exponents, CRT-RSA is less secure when the decryption exponents are of different bit size than the case when they are of same bit size.

In Table 3 we present the numerical results for partial key exposure attack. We consider $d_p < d_q$ and no information is available regarding the bits of d_p. Thus, we have $\gamma_1 = \delta_1$ and $(\delta_2 - \gamma_2)\log_2 N$ MSBs of d_q need to be known for the attack. Moreover, we consider two cases: (i) when no information regarding p is known and (ii) when $0.01\log_2 N$ MSBs of p are known. As a particular instance, when $d_p < N^{0.06}$, then one may attack CRT-RSA with $d_q < N^{0.097}$,

Table 3. Numerical results following Theorem 3

δ_1	δ_2	γ_2, when $\beta = \frac{1}{2}$	γ_2, when $\beta = \frac{1}{2} - 0.01$
0.03	0.136	0.102	0.122
0.04	0.121	0.091	0.107
0.05	0.109	0.078	0.093
0.06	0.097	0.068	0.082

when $(0.097 - 0.082)\log_2 N = 0.015\log_2 N$ MSBs of d_q are exposed and also $0.01\log_2 N$ MSBs of p is available.

For experimental results, one needs to use limited lattice dimensions and it may not be possible to reach these bounds in practice.

4 Conclusion

Using the idea of [16], we have studied the cryptanalysis of CRT-RSA when certain amount of the MSBs of the decryption exponents d_p, d_q are exposed. The attack becomes sharper with the knowledge of a few MSBs of p. The results work for any e of $O(N)$ and primes of the same bit size. Our results demonstrate that the upper bounds of insecure decryption exponents increase with the exposure of certain amounts of their MSBs. We also study the case when the decryption exponents are of different bit size. Our results show that CRT-RSA is more insecure in this case (considering the sum of bits in the decryption exponents) than when the decryption exponents are of the same bit size.

Acknowledgments. The authors like to thank the anonymous reviewers for detailed comments that improved the technical as well as editorial quality of this paper. The first author likes to acknowledge the Council of Scientific and Industrial Research (CSIR), India for supporting his research fellowship.

References

1. Bleichenbacher, D., May, A.: New Attacks on RSA with Small Secret CRT-Exponents. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 1–13. Springer, Heidelberg (2006)
2. Blömer, J., May, A.: New Partial Key Exposure Attacks on RSA. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 27–43. Springer, Heidelberg (2003)
3. Boneh, D., Durfee, G., Frankel, Y.: Exposing an RSA Private Key Given a Small Fraction of its Bits. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 25–34. Springer, Heidelberg (1998)
4. Boneh, D.: Twenty Years of Attacks on the RSA Cryptosystem. Notices of the AMS 46(2), 203–213 (1999)
5. Boneh, D., Durfee, G.: Cryptanalysis of RSA with Private Key d Less Than $N^{0.292}$. IEEE Trans. on Information Theory 46(4), 1339–1349 (2000)
6. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. Journal of Cryptology 14(2), 101–119 (2001)

7. Cohen, H.: A Course in Computational Algebraic Number Theory. Springer, Heidelberg (1996)
8. Coppersmith, D.: Small Solutions to Polynomial Equations and Low Exponent Vulnerabilities. Journal of Cryptology 10(4), 223–260 (1997)
9. Coron, J.-S.: Finding Small Roots of Bivariate Integer Equations Revisited. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 492–505. Springer, Heidelberg (2004)
10. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms, 2nd edn. Springer, Heidelberg (1998)
11. Ernst, M., Jochemsz, E., May, A., de Weger, B.: Partial Key Exposure Attacks on RSA up to Full Size Exponents. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 371–386. Springer, Heidelberg (2005)
12. Galbraith, S., Heneghan, C., Mckee, J.: Tunable Balancing of RSA. In: Boyd, C., González Nieto, J.M. (eds.) ACISP 2005. LNCS, vol. 3574, pp. 280–292. Springer, Heidelberg (2005)
13. Howgrave-Graham, N.: Finding Small Roots of Univariate Modular Equations Revisited. In: Darnell, M.J. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 131–142. Springer, Heidelberg (1997)
14. Jochemsz, E.: Cryptanalysis of RSA Variants Using Small Roots of Polynomials. Ph. D. thesis, Technische Universiteit Eindhoven (2007)
15. Jochemsz, E., May, A.: A Strategy for Finding Roots of Multivariate Polynomials with new Applications in Attacking RSA Variants. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 267–282. Springer, Heidelberg (2006)
16. Jochemsz, E., May, A.: A Polynomial Time Attack on RSA with Private CRT-Exponents Smaller Than $N^{0.073}$. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 395–411. Springer, Heidelberg (2007)
17. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring Polynomials with Rational Coefficients. Mathematische Annalen 261, 513–534 (1982)
18. May, A.: Cryptanalysis of Unbalanced RSA with Small CRT-Exponent. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 242–256. Springer, Heidelberg (2002)
19. May, A.: Using LLL-Reduction for Solving RSA and Factorization Problems: A Survey. LLL+25 Conference in honour of the 25th birthday of the LLL algorithm (2007), http://www.informatik.tu-darmstadt.de/KP/alex.html (last accessed 23 December, 2008)
20. Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Communications of ACM 21(2), 158–164 (1978)
21. Wiener, M.: Cryptanalysis of Short RSA Secret Exponents. IEEE Transactions on Information Theory 36(3), 553–558 (1990)

How to Compare Profiled Side-Channel Attacks?

François-Xavier Standaert[1,⋆], François Koeune[1,⋆⋆], and Werner Schindler[2]

[1] UCL Crypto Group, Université catholique de Louvain, B-1348 Louvain-la-Neuve
{fstandae,francois.koeune}@uclouvain.be

[2] Bundesamt für Sicherheit in der Informationstecknik (BSI), 53175 Bonn, Germany
werner.schindler@bsi.bund.de

Abstract. Side-channel attacks are an important class of attacks against cryptographic devices and profiled side-channel attacks are the most powerful type of side-channel attacks. In this scenario, an adversary first uses a device under his control in order to build a good leakage model. Then, he takes advantage of this leakage model to exploit the actual leakages of a similar target device and perform a key recovery. Since such attacks are divided in two phases (namely profiling and online attack), the question of how to best evaluate those two phases arises. In this paper, we take advantage of a recently introduced framework for the analysis of side-channel attacks to tackle this issue. We show that the quality of a profiling phase is nicely captured by an information theoretic metric. By contrast, the effectiveness of the online key recovery phase is better measured with a security metric. As an illustration, we use this methodology to compare the two main techniques for profiled side-channel attacks, namely template attacks and stochastic models. Our results confirm the higher profiling efficiency of stochastic models when reasonable assumptions can be made about the leakages of a device.

1 Introduction

Side-channel attacks are a powerful class of cryptanalysis techniques in which an adversary not only takes advantage of the mathematical properties of an algorithm but also of the physical properties of its implementation. Profiled side-channel attacks are the most powerful type of side-channel attacks and can be viewed as divided in two phases. First, a profiling phase provides an adversary with a training device and allows him characterizing its physical leakages. Second, an online exploitation phase is mounted against a similar target device in order to perform a key recovery. Standard profiled side-channel attacks include template attacks and stochastic models, respectively introduced in [1] and [5].

Because of their division in two phases, a usual question for such attacks is to determine their effectiveness in profiling and attacking a device. In this work, we follow the analysis of [2] in which the performances of template attacks and stochastic models were analyzed. In this reference, the efficiency of

[⋆] Associate researcher of the Belgian Fund for Scientific Research (FNRS - F.R.S.)

[⋆⋆] Supported in part by the Walloon Region research project SCEPTIC.

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 485–498, 2009.

the online phase was nicely captured by measuring the success rate of a key recovery adversary exploiting templates or stochastic models. By contrast, the criteria used to quantify the quality of the profiling phase had a more ad hoc flavor. As a consequence, we suggest that the framework of [7] can be used to improve this analysis. We present experiments to confirm how and why an information theoretic metric captures the profiling efficiency of an attack while a security metric rather measures the effectiveness of its online phase. Hence, our results confirm the previous intuitions with a more rigorous theoretical background. In practice, we observe that stochastic models built from sound engineering assumptions can give a very precise image of a device's leakages from a reduced amount of profiling measurements. More formally, our experiments can be viewed as the practical counterparts of Theorems 1 and 2 in [7]. They show that the proposed principles for comparing side-channel attacks are not only theoretical but can also be practically meaningful and solve actual engineering problems.

The rest of this paper is structured as follows. Section 2 introduces the preliminary assumptions in profiled side-channel attacks. Section 3 recalls the evaluation metrics of [7]. Section 4 provides a brief description of the template attacks and stochastic models with a discussion of their parameters. The core of the paper is in Sections 5 and 6 in which our experimental comparisons are presented and their limitations are analyzed. Eventually, conclusions are in Section 7.

2 Preliminary Assumptions for Profiled Attacks

Before starting a careful analysis of particular types of attacks, it is important to consider the different assumptions that can sometimes be hidden in the description and implementation of a profiled side-channel attack. In particular, this section aims to list four decisions that generally have to be taken.

Known or chosen plaintext models. As a matter of fact, any profiled side-channel attack starts by building a leakage model that will be used in the online part of an attack to predict the actual leakages of a target device. As a consequence arises the question: "for which inputs will the model be built?". In the practice of side-channel attacks, there are essentially two available choices, namely known or chosen plaintext leakage models. If a *chosen plaintext leakage model* is decided, it suffices if the adversary only builds a model for certain plaintexts or sequences of plaintexts. Hence, the same chosen plaintexts or sequences of plaintexts will have to be used in the online phase of the attack. By contrast, if a *known plaintext leakage model* is considered, the leakages corresponding to the encryption of any plaintext can be exploited in the online phase of the attack[1].

[1] We mention that chosen plaintext *models* are not particularly desirable in template attacks (since they limit the exploitable plaintexts in the online phase of the attack). But they should not be confused with (possibly adaptive) chosen plaintext *attacks* that generally improve the effectiveness of the online phases. Note also that known or chosen ciphertexts could be considered equivalently.

Weight or distance based models. A side-channel adversary always has to do some minimal assumptions on the architecture of his target device. Typically, side-channel leakages such as the power consumption are generally dependent on the transition between two inputs rather than on single inputs. If such transition-based leakages are actually observed, it implies that the models also have to be built for different input transitions rather than for different inputs. Such a context typically corresponds to the Hamming distance leakage models described, *e.g.* in [4]. By contrast, in certain devices (*e.g.* smart cards) the meaningful transitions are not between two inputs but between a variable input and a constant state. In such scenarios, leakage models based on single inputs are again meaningful, just as when Hamming weight models apply[2].

Symmetry properties in the leakages. Depending on the two previous decisions, an adversary will decide to build a model (*e.g.* templates) for different inputs of the target device. In the context of a block cipher, it means that models have to depend on plaintexts and keys. But the lower the number of templates to build, the better the profiling efficiency. Hence, one will typically try to take advantage of symmetry properties in the leakages, such as the Equal Images under different Subkeys (EIS) property defined in [5]. For example, if it is known that (most of) the leakages of a block cipher implementation are not dependent on both the plaintext and the key but only on the XOR between the plaintext and the key, then templates can be built only for these XOR values. For further considerations on symmetries, we refer the interested reader to [6].

Need to program a target device. Eventually, it is worth mentioning that it is generally assumed that profiled side-channel attacks require a device that one can program (*e.g.* control the keys) during profiling. In fact, if an EIS property is assumed, it can be sufficient to profile the device with only one known key. When stochastic models are considered, it may even be possible to profile without a device for which the key is known (see [5], Remark 2 for the details).

2.1 Target Implementation

The goal of this paper is not to investigate one particular device but to provide a methodological contribution to the comparison of profiled side-channel attacks. For this reason, we decided to analyze a simple simulated attack scenario in which all the parameters are under control. As will be clear later, it allows putting forward interesting intuitions on the respective effectiveness of the template attacks and stochastic models but also on their limitations.

In practice, we investigated the following context. Let k be the first master key byte of the AES Rijndael and x_i be a corresponding input plaintext byte. Let S be the AES S-box and $y_i = \mathsf{S}(x_i \oplus k)$ be the output of this S-box. We consider an

[2] In theory, longer history effects could be observed, *i.e.* the actual leakages may not only depend on the transition between two inputs but also on previous ones. We focus on the weight and distance based models because they are very common in the literature. But extending the choice towards other cases would be possible.

adversary that is provided with leakage traces[3] of the form $[x_i, H_W(\mathsf{S}(x_i \oplus k)) + n_i]$ where H_W is the Hamming weight function and n_i is a realization of normally distributed noise, described by a random variable N_i with expectation $\mu = 0$ and with variance σ^2. In the following sections, we will evaluate this adversary in function of two parameters: the amount of traces used in the profiling stage of the attack q_p and the amount of traces used in the online phase of the attack q. With respect to the previous assumptions, we will build known plaintext models assuming weight based leakages. Eventually, the adversary will take advantage of an EIS property and assume that the leakage for every pair (x_1, k_1), (x_2, k_2) such that $x_1 \oplus k_1 = x_2 \oplus k_2$ is identical. We acknowledge that this scenario (mainly selected for tutorial purposes) hides the practical problem of selecting the meaningful time samples in the leakage traces (discussed, *e.g.* in [2,9]), due to its univariate nature. However, the proposed evaluation methodology can be straightforwardly extended to multivariate probability distributions.

3 Evaluation Metrics

Following the framework introduced in [7], we will evaluate our different experiments with a combination of information theoretic and security metrics.

Information theoretic metric. Let K be a discrete random variable representing the target key byte of our side-channel attacks and k be a realization of this variable (*i.e.* the key in one instance of attack). Let $\mathbf{L}_q$ be a random vector describing random side-channel observations generated with q queries to the target physical computer and $\mathbf{l}_q = [l_1, l_2, \ldots, l_q]$ be a realization of this random vector, with *e.g.* $l_i = H_W(\mathsf{S}(x_i \oplus k)) + n_i$ (and $L_i = H_W(\mathsf{S}(x_i \oplus k)) + N_i$) as explained in the previous section. Let finally $\Pr[k|\mathbf{l}_q]$ be the conditional probability of a key byte k given a leakage $\mathbf{l}_q$. We define a conditional entropy matrix as:

$$\mathbf{H}^q_{k,k^*} = -\sum_{\mathbf{l}_q} \Pr[\mathbf{l}_q|k] \cdot \log_2 \Pr[k^*|\mathbf{l}_q], \tag{1}$$

where k^* denotes a possible key class candidate in the attack. From this matrix, we derive Shannon's conditional entropy as follows:

$$\mathrm{H}[K|\mathbf{L}_q] = -\sum_{k} \Pr[k] \sum_{\mathbf{l}_q} \Pr[\mathbf{l}_q|k] \cdot \log_2 \Pr[k|\mathbf{l}_q] = \mathop{\mathbf{E}}_{k} \; \mathbf{H}^q_{k,k},$$

where $\mathbf{E}$ denotes the mathematical expectation and $\Pr[k|\mathbf{l}_q]$ is derived from the Bayes law. We note that this definition is equivalent to the classical one since:

$$\begin{aligned}\mathrm{H}[K|\mathbf{L}_q] &= -\sum_{\mathbf{l}_q} \Pr[\mathbf{l}_q] \sum_{k} \Pr[k|\mathbf{l}_q] \cdot \log_2 \Pr[k|\mathbf{l}_q] \\ &= -\sum_{k} \Pr[k] \sum_{\mathbf{l}_q} \Pr[\mathbf{l}_q|k] \cdot \log_2 \Pr[k|\mathbf{l}_q]\end{aligned}$$

[3] Each trace contains only one leakage sample, *i.e.* we only consider univariate attacks.

Then, we define an entropy reduction matrix: $\widetilde{\mathbf{H}}^q_{k,k^*} = \mathrm{H}[K] - \mathbf{H}^q_{k,k^*}$, where $\mathrm{H}[K]$ is the entropy of the key byte K before any side-channel attack has been performed: $\mathrm{H}[K] = -\mathbf{E}_k \log_2 \Pr[k]$. It directly yields the mutual information:

$$\mathrm{I}(K;\mathbf{L}_q) = \mathrm{H}[K] - \mathrm{H}[K|\mathbf{L}_q] = \mathop{\mathbf{E}}_k \; \widetilde{\mathbf{H}}^q_{k,k} \tag{2}$$

Security metric. We consider a side-channel key recovery adversary of which the aim is to guess a key byte k with non negligible probability. For this purpose and for each candidate k^*, he compares the actual observation of a leaking device $\mathbf{l}_q$ with some key dependent model for these leakages $\mathsf{M}(k^*,.)$. The construction of these models (otherwise said templates or stochastic models) will be detailed in the next section. Let $\mathsf{T}(\mathbf{l}_q, \mathsf{M}(k^*,.))$ be the statistical test used in the comparison. We assume that the highest value of the statistic corresponds to the most likely key candidate. For each observation $\mathbf{l}_q$, we store the result of the statistical test T in a vector $\mathbf{g}_q = \mathsf{T}(\mathbf{l}_q, \mathsf{M}(k^*,.))$ containing the key candidates sorted according to their likelihood: $\mathbf{g}_q := [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ (*e.g.* in our present context $|\mathcal{K}|$=256). Then, for any side-channel attack exploiting a leakage vector $\mathbf{l}_q$ and giving rise to a result $\mathbf{g}_q$, we define the success function of order o against a key byte k as: $\mathsf{S}^o_k(\mathbf{g}_q)$=1 if $k \in [g_1, \ldots, g_o]$, else $\mathsf{S}^o_k(\mathbf{g}_q)$=0. It leads to the o^{th}-order success rate:

$$\mathbf{Succ}^o_K = \mathop{\mathbf{E}}_k \; \mathop{\mathbf{E}}_{\mathbf{l}_q} \; \mathsf{S}^o_k(\mathbf{g}_q) \tag{3}$$

Intuitively, a success rate of order 1 (*resp.* 2) relates to the probability that the correct key byte is sorted first (*resp.* among the two first ones) by the adversary.

4 Description of the Attacks

4.1 Classical Template Attacks

Templates construction. Suppose that an adversary is provided with N_x leakage traces corresponding to the computation of a secret value v. As will be discussed in Section 4.3, this value can but does not have to be the secret key k. In theory, one can build templates for any intermediate value computed by a leaking cryptographic device. In the template attacks of [1], a multivariate Gaussian noise is considered, which means that the vectors $\{\mathbf{l}^{v,i}_q\}^{N_x}_{i=1}$ are assumed to be drawn from the multivariate distribution:

$$\mathcal{N}(\mathbf{l}^{v,i}_q|\boldsymbol{\mu}_v, \boldsymbol{\Sigma}_v) = \frac{1}{(2\pi)^{\frac{N}{2}}|\boldsymbol{\Sigma}_v|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(\mathbf{l}^{v,i}_q - \boldsymbol{\mu}_v)^\top \boldsymbol{\Sigma}_v^{-1}(\mathbf{l}^{v,i}_q - \boldsymbol{\mu}_v)\right\},$$

where the mean $\boldsymbol{\mu}_v$ and the covariance matrix $\boldsymbol{\Sigma}_v$ specify completely the noise distribution associated to each secret v. Constructing the templates consists then in estimating the sets of parameters $\{\boldsymbol{\mu}_v\}^{|\mathcal{V}|}_{v=1}$ and $\{\boldsymbol{\Sigma}_v\}^{|\mathcal{V}|}_{v=1}$. A standard approach is to use the empirical mean and covariance matrix associated to the observations $\{\mathbf{l}^{v,i}_q\}^{N_x}_{i=1}$: $\hat{\boldsymbol{\mu}}_v = \frac{1}{N_x}\sum^{N_x}_{i=1} \mathbf{l}^{v,i}_q$, $\widehat{\boldsymbol{\Sigma}}_v = \frac{1}{N_x}\sum^{N_x}_{i=1}(\mathbf{l}^{v,i}_q - \hat{\boldsymbol{\mu}}_v)(\mathbf{l}^{v,i}_q - \hat{\boldsymbol{\mu}}_v)^\top$.

Attack. Assume now that there are $|\mathcal{V}|$ possible secret values. In order to determine by which secret signal a new vector $\mathbf{l}_{\text{new}}$ was generated, we apply Bayes' rule. This leads to the following classification rule:

$$\tilde{v} = \underset{v^*}{\text{argmax}}\ \hat{\Pr}[v^*|\mathbf{l}_{\text{new}}] = \underset{v^*}{\text{argmax}}\ \hat{\Pr}[\mathbf{l}_{\text{new}}|v^*] \Pr[v^*],$$

where $\hat{\Pr}[\mathbf{l}_{\text{new}}|v^*] = \mathcal{N}(\mathbf{l}_{\text{new}}|\hat{\boldsymbol{\mu}}_{v^*}, \widehat{\boldsymbol{\Sigma}}_{v^*})$ and $\Pr[v^*]$ is the a priori probability of the value candidate v^*. The classification rule assigns $\mathbf{l}_{\text{new}}$ to the candidate v^* with the highest a posteriori probability. In general, we have $\Pr[v^*] = \frac{1}{|\mathcal{V}|}$.

Interestingly, such template attacks require N_x traces to build each of the $|\mathcal{V}|$ possible models (*i.e.* mean vectors, covariance matrices). Hence, the overall number of traces for profiling q_p equals $N_x \times |\mathcal{V}|$. We note again that in our example, each execution of the S-box only gives rise to a single leakage sample. Hence we are limited to univariate attacks. But the following analysis would apply identically if each leakage trace was containing several samples.

Finally, in the (frequent) case where the values v for which the templates are built are not equal to the target key k, the adversary additionally combines the leakages corresponding to different key-dependent values in order to perform a key recovery, *i.e.* he computes $\tilde{k} = \underset{k^*}{\text{argmax}}\ \prod_{i=1}^{q} \hat{\Pr}[l_{\text{new},i}|x_i, k^*]$.

4.2 Stochastic Models

The stochastic models introduced in [5] work in a slightly different fashion than classical template attacks in the sense that they attempt to take advantage of the adversary's knowledge of the target device during the profiling phase. Let $\mathbf{l}_q = [l_1, l_2, \ldots, l_q]$ be the leakage vector defined in the previous sections, l_i be a leakage trace and $l_i(t)$ a leakage sample in this trace. In theory, any of those samples is the output of a leakage function L_t such that, *e.g.* in our block cipher context, $l_i(t) = \mathsf{L}_t(x_i, k)$. Stochastic models assume that this leakage function can be written as the sum of a deterministic part and a random part, namely: $\mathsf{L}_t(x_i, k) = \delta_t(x_i, k) + \rho_t$. From this basic assumption results the fact that the profiling phase will now be divided in two parts in order to approximate the leakage function deterministic part and random part separately.

Approximation of the leakage function deterministic part. In this first phase, it is assumed that the deterministic part of the leakage function can be approached as a linear combination $\hat{\delta}_t(x_i, k) = \sum_{j=0}^{u-1} \beta_{j,t} \cdot g_{j,t}(x_i, k)$, for some well chosen base functions $g_{j,t}$ of the plaintext and the key[4]. Hence, the goal of this first phase is to find the closest approximation of this form. Finding a good base $[g_{0,t}, g_{1,t}, \ldots, g_{u-1,t}]$ is typically where engineering intuition can be exploited since one has to select the functions of which the output influences the actual leakages. The better the base vector functions are correlated with the actual leakages, the better the approximation of δ_t. Quite naturally, the best situation

[4] ... and any other possible input, *e.g.* the masks in case of protected designs.

for an adversary is to have a small basis that perfectly captures all the leakage dependencies, *i.e.* to have a fast convergence towards a good approximation.

In practice, the adversary first generates N_1 leakage traces corresponding to plaintexts x_i and keys k and builds the following matrix:

$$A = \begin{pmatrix} g_{0,t}(x_1,k) & g_{1,t}(x_1,k) & ... & g_{u-1,t}(x_1,k) \\ g_{0,t}(x_2,k) & g_{1,t}(x_2,k) & ... & g_{u-1,t}(x_2,k) \\ ... & ... & ... & ... \\ g_{0,t}(x_{N_1},k) & g_{1,t}(x_{N_1},k) & ... & g_{u-1,t}(x_{N_1},k) \end{pmatrix}$$

As mentioned in Section 2, depending on the exploitation or not of a symmetry property in the leakages, it can be necessary or not to actually change the key during the profiling (note that is generally true for template attacks as well). Then, the adversary takes the leakage vector $\mathbf{l}_{N_1}(t) = [l_1(t), l_2(t), \ldots, l_{N_1}(t)]$ corresponding to the encryption of the same plaintexts with the same keys as in the matrix A. The approximation of δ_t can eventually be obtained by applying the least square method and simply computing the coefficients $\beta_{j,t}$ as follows:

$$\mathbf{b}_t = [\beta_{0,t}, \beta_{1,t}, \ldots, \beta_{u-1,t}] = (A^T \cdot A)^{-1} \cdot A^T \cdot \mathbf{l}_{N_1}(t)$$

Approximation of the leakage function random part. As for the previous template attacks, stochastic models assume a multivariate gaussian distribution for the random part of the leakages. In order to approximate this distribution, the adversary generates N_2 new traces and first evaluates a random vector that corresponds to the approximation error for m different time samples:

$$\mathbf{r}_m = [r_{t_1}, r_{t_2}, \ldots, r_{t_m}], \text{ with } r_{t_j} = \mathsf{L}_{t_j}(x_i,k) - \hat{\delta}_{t_j}(x_i,k)$$

From the N_2 realizations of the corresponding random variable R_m, he then computes the $m \times m$ empirical covariance matrix C such that $c_{ij} = Cov(r_{t_i}, r_{t_j})$.

Attack. In this third phase, the adversary obtains N_3 new traces $l_{\text{new},i}{=}\mathsf{L}(x_i,k)$. For each of those traces, he first computes a noise vector: $\mathbf{z}_i = [l_{\text{new},i}(t_1) - \hat{\delta}_{t_1}(x_i,k), l_{\text{new},i}(t_2) - \hat{\delta}_{t_2}(x_i,k), \ldots, l_{\text{new},i}(t_m) - \hat{\delta}_{t_m}(x_i,k)]$. From this vector, he can compute the following probabilities:

$$\hat{\Pr}[\mathbf{z}_i|x_i,k^*] = \frac{1}{\sqrt{(2\pi)^m|C|}} \exp\left\{-\frac{1}{2}\mathbf{z}_i^T C^{-1}\mathbf{z}_i\right\}$$

Finally, he combines these probabilities and applies the maximum likelihood rule:

$$\tilde{k} = \underset{k^*}{\operatorname{argmax}} \prod_{i=1}^{N_3} \hat{\Pr}[\mathbf{z}_i|x_i,k^*]$$

Hence, the total number of traces for profiling a stochastic model equals $q_p = N_1 + N_2$ and the number of traces in the online phase equals $q = N_3$.

4.3 Selection of Templates and Base Vectors

A consequence of the previous descriptions is that both template attacks and stochastic models need to do some arbitrary choices before starting to profile a device. In the context of template attacks, one has to define the secret values v for which templates will be built. When stochastic models are considered, one has to determine the base functions. Therefore, if our goal is to compare those techniques on a fair basis, it is important to perform this arbitrary choice with assumptions as close as possible. For the template attacks, because we assume an EIS property, we decided to build templates for each of the $|\mathcal{V}| = 256$ possible values of $x_i \oplus k$. For the stochastic models, we assumed that the leakages were dependent of the 8 bits of the S-box output $y_i = \mathsf{S}(x_i \oplus k)$. Hence the base functions used in our experiments were $[1, y_i(1), y_i(2), \ldots, y_i(8)]$, where $y_i(j)$ denotes the j^{th} bit of y_i (here interpreted as a real number). We note that for both attacks, we could similarly assume that the leakages only depend on the Hamming weight of the S-box output. It would have resulted in the construction of only 9 templates corresponding to those Hamming weights and the use of a 2-dimensional basis $[1, H_W(y_i)]$ for the stochastic models.

5 Experiments

5.1 Empirical Computation of the Metrics

In this section, we present the results of different simulated profiled attacks. For this purpose, we empirically evaluated our different metrics as follows.

1. We generated large amounts of profiling traces.
2. We split these traces in different sets of q_p traces (with $N_1 = N_2 = q_p/2$)[5].
3. For various q_p values, we constructed templates and stochastic models.
4. Eventually and for various number of traces q, we evaluated the attacks, *i.e.*
   - We evaluated the probabilities $\hat{\Pr}[k^*|x_i, l_{\text{new},i}]$,
   - From those probabilities, we estimated the first-order success rate in function of q and the conditional entropy $\hat{\mathrm{H}}[K|\mathbf{L}_1]$.

In practice, the traces were generated from uniformly distributed plaintexts. We mention that since all our experiments are simulated, we were not limited in the amount of traces generated nor by statistical sampling problems in the estimation of the metrics. Note also that, following the analysis in [7], the success rate was estimated in function of the number of queries in the online phase of the attacks. By contrast, the conditional entropy was only estimated for $q = 1$.

[5] Usually, N_2 should increase as the number m of considered time instants $t_1, \ldots, t_m$ increases, while m is irrelevant for the choice of N_1. Also, if the implementation has no symmetries, N_2 is generally of subordinate relevance compared to N_1.

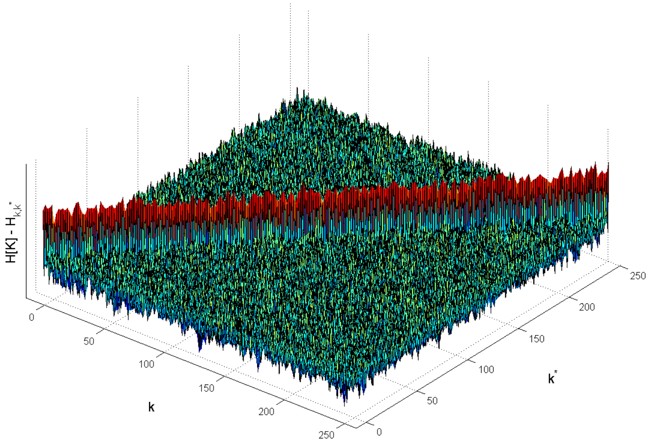


Fig. 1. Entropy reduction matrix of a sound leakage model

5.2 Sanity Check: The Conditional Entropy Matrix

A first interesting step in the evaluation of a leakage model (*i.e.* templates-based or stochastic) is to check if it is at least good enough to perform a successful key recovery. The conditional entropy matrix is a particularly useful tool with this respect. As demonstrated in [7], Theorem 1, a matrix $\hat{\mathbf{H}}^1_{k,k*}$ such that its diagonal values are minimum for all keys indicates a sound leakage model (*i.e.* a leakage model that allows asymptotically successful key recoveries). Hence, any time we constructed a new leakage model, we checked its soundness. For example, Figure 1 illustrates the entropy reduction matrix of a sound leakage model obtained from a template attack in which every template was profiled with 16 traces. We can clearly see the significantly higher information leakages of the diagonal values. It is interesting to observe that while a sound leakage model guarantees a successful key recovery, it is not a necessary condition. One could easily imagine a leakage model such that only certain templates have been properly profiled and nevertheless leads to successful attacks.

5.3 Evaluation of the Attacks

Next to the sanity check of the conditional entropy matrix, Figures 2 and 3 respectively represent the estimation of our metrics for the two considered attacks. Interestingly, the success rate plot is 2-dimensional since it depends on both q_p and q. By contrast, the conditional entropy plot is only computed for $q = 1$ and hence only depends on q_p. Quite naturally, the success rate tends to one when the number of traces in the profiling and online phases increases. It is worth noting that the conditional entropy value is sometimes higher than 8 which clearly indicates an insufficient profiling. From a practical point of view, the figure directly suggests the increased effectiveness of the profiling phase when using stochastic models compared to template attacks. This is because only one function in a

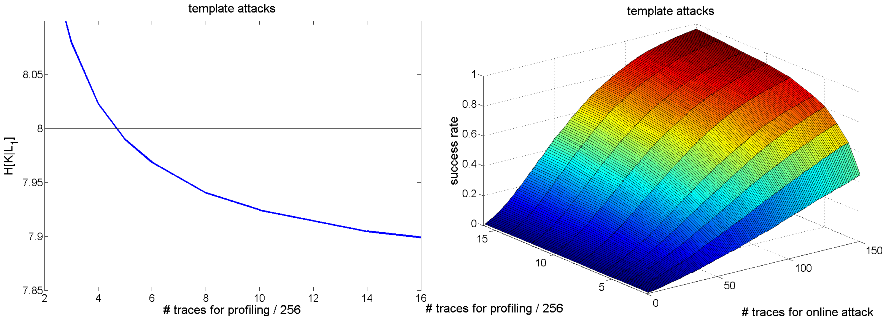


Fig. 2. Conditional entropy and success rate of the template attacks

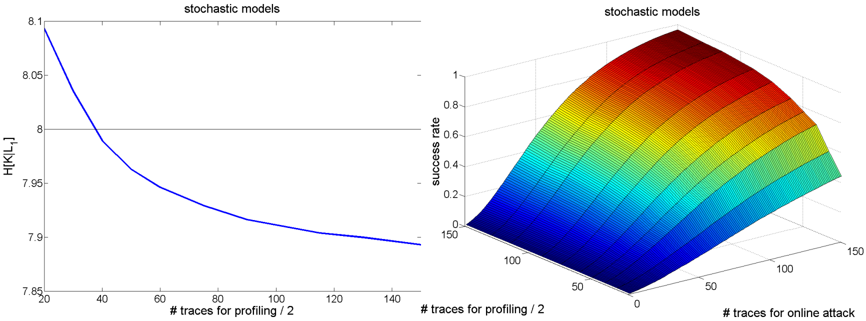


Fig. 3. Conditional entropy and success rate of the stochastic models

9-dimensional subspace has to be approximated compared to the building of 256 templates. From a more theoretical point of view, we can see that an increase in the amount of traces for profiling improves the effectiveness of the attacks (or informativeness of the models) up to a certain bound. It is consequently interesting to use the information theoretic metric to determine this bound. In our example, we can observe that template attacks and stochastic models have their conditional entropy that seem to converge towards the same value. It indicates that the base functions used to approximate the leakages properly capture their dependencies (which is expected since we know that the leakages actually correspond to the noisy Hamming weights of the AES S-box output).

5.4 Comparison of the Attacks

Given the previous results, a very natural question is to wonder if we can properly quantify the effectiveness of the profiling and online phases of the investigated attacks. As a matter of fact, this question can be divided in three parts: (1) "which profiling is the fastest to build a sound model?", (2) "which profiling gives rise to the smallest conditional entropy?" and (3) "which profiling gives rise to the most efficient online attacks?". In order to answer these questions, it is convenient to plot the conditional entropy values in a logarithmic scale

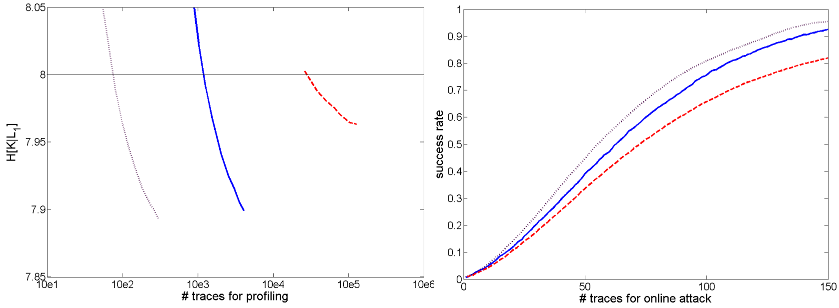


Fig. 4. Left figure: comparison of the conditional entropies - plain: template attacks, dotted: stochastic models, dashed: histograms. Right figure: comparison of the success rates - plain: template attack with $\mathrm{H}[S|\mathbf{L}_1] \simeq 7.94$, dotted: stochastic model with $\mathrm{H}[S|\mathbf{L}_1] \simeq 7.92$, dashed: stochastic model with $\mathrm{H}[S|\mathbf{L}_1] \simeq 7.98$.

as in the left part of Figure 4. From this picture, it clearly appears that the profiling of stochastic models is one order of magnitude faster than the one of classical template attacks in our example, which answers the first question. We then see (again) that both methods seem to converge towards the same conditional entropy value which answers the second question. Eventually and following [7], Theorem 2, this also implies that stochastic models and template attacks should be as efficient in the online attacks if a sufficient profiling is used. This is because a more informative model generally gives rise to a more efficient online attack. As an illustration, we plotted the success rates corresponding to three different profiling phases in the right part of the figure and they confirm this intuition. Hence, the information theoretic and security metrics appear as good methods for the comparison of the profiling and online attack efficiencies, respectively. In practice, since the main goal of a profiling step is to build a precise leakage model, the most important parameter to compare this step is usually the smallest value of the conditional entropy that can be reached. But when the limit of this conditional entropy for increasing q_p values is identical for different methods or in contexts where the number of profiling traces is limited, the rapidity of converging towards a sound leakage model becomes important as well.

Summarizing, the speed of convergence of a profiling method is measured by the X axis in the left part of Figure 4; the informativeness of the profiled models is measured by the Y axis of Figure 4; and this informativeness is generally related to the success rate of the corresponding online attacks.

We mention that for illustration, we also evaluated a naive profiling in which the Gaussian templates were replaced by histograms. As observed in the left part of Figure 4, such histograms are slower to build less informative models. In theory, one could of course imagine many other types and contexts of profiling (*e.g.* profiling that produces sound but not very informative models very fast or profiling that produces very informative models very slowly).

6 Limitations

The previous sections were dedicated to the description of an exemplary context in which the proposed methodology to compare profiled side-channel attacks was meaningful. Before concluding the paper, this section aims to briefly discuss the extent to which the previous conclusions are generally true.

A first restriction that has to be mentioned relates to the evaluation framework itself. As demonstrated in [7], there is no one-to-one relation between the conditional entropy and the success rate computed for a general leakage function. In numerous practical applications, the intuition that more conditional entropy implies less success rate is verified. But this does not prevent the possible existence of counterintuitive situations. It remains that the proposed metrics and relations are at least more meaningful than ad hoc evaluation criteria. But a certain level of scepticism and the verification of some relations such as in the right part of Figure 4 are always in place in the analysis of side-channel attacks.

A second restriction relates to the evaluation of the metrics in real measurement environments where statistical sampling can become an issue. As a matter of fact, reaching a high confidence level in the evaluation of the metrics when computed from small unprotected devices is generally not an issue. But, *e.g.* computing the conditional entropy for a protected hardware design can be more difficult. With this respect, it is worth remembering that comparing implementations according to their information leakages is only meaningful in the context of sound leakage models. Hence, the more challenging the target device, the more interesting the entropy matrix sanity check of Section 5.2.

Thirdly, it is important to acknowledge that the comparison between two side-channel attacks such as templates attacks and stochastic models in this paper is in essence implementation-dependent. What this paper provides is a methodology that allows comparing these attacks on a sound basis, for one given implementation (or for a class of similar implementations). But changing the experimental conditions can affect the practical conclusions that are obtained from a set of experiments. For example, we conclude from our investigated context that the profiling efficiency of stochastic models is much higher than the one of classical template attacks. In fact, this conclusion mainly holds because the base functions chosen to build our stochastic models perfectly capture the actual leakage dependencies. But in case the base vectors are not perfectly chosen, the generic nature of template attacks may allow them to better incorporate the physical specificities of the measurements. Yet, we point out that the subspace can always be selected so large that it catches all relevant peculiarities, possibly at cost of profiling efficiency. Hence, template attacks can be viewed as the limiting case of the stochastic approach, when the subspace equals the full vector space. In other words, stochastic models generally trade a bit of the generality of template attacks for a more efficient (*i.e.* faster) profiling.

Eventually, let us mention that there are situations where templates are more appropriate than stochastic models. An interesting example is the following. Say the S-box in a block cipher is unknown and a device only leaks the Hamming weights of this S-box output. Then, templates can still be built for any value of

$x_i \oplus k$ and result in a sound leakage model. By contrast, the previous (standard) selection of basis vectors that depend on $\mathsf{S}(x_i \oplus k)$ is not possible anymore. And a basis made of the 8 bits of $x_i \oplus k$ will not lead to a good approximation of the leakage function, because it does not not capture the S-box non-linearity.

7 Conclusions

This paper presents an application of the methodology introduced in [7] to the analysis of template attacks and stochastic models. We investigated an exemplary context of simulated leakages in order to confirm the soundness of some metrics to compare profiled side-channel attacks. Extending this analysis and evaluation towards more complex scenarios is a good scope for further research.

In particular, the evaluation of multivariate attacks against masked implementations or non-CMOS devices would be interesting. Since in general, the problem of power-based side-channel attacks can be viewed as a probability density function estimation problem, it is expected that the intuition provided by an information theoretic analysis as in this work will generally hold. But additional empirical confirmations would strengthen this expectation. For example, it is known that the conditional entropy can be used to evaluate masked implementations [8] and that stochastic models are also applicable in this context [3,6]. A practical question is to determine how much the masking exactly affects the profiling efficiency of profiled attacks (with known or unknown masks).

Acknowledgements. The authors would like to thank the reviewers of ACNS 2009 for their meaningful comments about this work.

References

1. 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)
2. Gierlichs, B., Lemke, K., Paar, C.: Templates vs. Stochastic Methods. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 15–29. Springer, Heidelberg (2006)
3. Lemke, K., Paar, C.: Analyzing Side-Channel Leakage of Masked Implementations with Stochastic Methods. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 454–468. Springer, Heidelberg (2007)
4. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks. Springer, Heidelberg (2007)
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. Schindler, W.: Advanced Stochastic Methods in Side-Channel Analysis on Block Ciphers in the Presence of Masking. J. of Math. Cryptology 2, 291–310 (2008)
7. Standaert, F.-X., Malkin, T.G., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. To appear in the proceedings of Eurocrypt (2009); Extended version available from: Cryptology ePrint Archive, Report 2006/139

8. Standaert, F.-X., Peeters, E., Archambeau, C., Quisquater, J.-J.: Towards Security Limits in Side-Channel Attacks. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 30–45. Springer, Heidelberg (2006), http://eprint.iacr.org/2007/222
9. Standaert, F.-X., Archambeau, C.: Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 411–425. Springer, Heidelberg (2008)

Theoretical and Practical Aspects of Mutual Information Based Side Channel Analysis

Emmanuel Prouff[2] and Matthieu Rivain[1,2]

[1] University of Luxembourg
[2] Oberthur Technologies
{e.prouff,m.rivain}@oberthur.com

Abstract. A large variety of side channel analyses performed on embedded devices involve the linear correlation coefficient as wrong-key distinguisher. This coefficient is actually a sound statistical tool to quantify linear dependencies between univariate variables. However, when those dependencies are non-linear, the correlation coefficient stops being pertinent so that another statistical tool must be investigated. Recent works showed that the *Mutual Information* measure is a promising candidate, since it detects any kind of statistical dependency. Substituting it for the correlation coefficient may therefore be considered as a natural extension of the existing attacks. Nevertheless, the first applications published at CHES 2008 have revealed several limitations of the approach and have raised several questions. In this paper, an in-depth analysis of side channel attacks involving the mutual information is conducted. We expose their theoretical foundations and we assess their limitations and assets. Also, we generalize them to higher orders where they seem to be an efficient alternative to the existing attacks. Eventually, we provide simulations and practical experiments that validate our theoretical analyses.

1 Introduction

Side Channel Analysis (SCA) is a cryptanalytic technique that consists in analyzing the physical leakage produced during the execution of a cryptographic algorithm embedded on a physical device. This side channel leakage is indeed statistically dependent on the intermediate variables of the computation which enables key recovery attacks.

Since their introduction in the nineties, several kinds of SCA have been proposed which essentially differ in the involved distinguisher. A first family is composed of SCA based on linear correlation distinguishers. When such an attack is performed, the adversary implicitly assumes that there is a linear dependence between its predictions and the leakage measurements. Actually, the attack effectiveness depends on the accuracy of this assumption. The most well-known examples of such attacks are the *Differential Power Analysis* (DPA) [1] that is based on a Boolean correlation and the *Correlation Power Analysis* (CPA) [2] that involves *Pearson correlation* coefficient. The second important family of SCA is composed of the so-called *Template Attacks* (TA) [3]. They involve

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 499–518, 2009.

maximum-likelihood distinguishers and can succeed when the DPA or CPA do not. However, TA can only be performed if the attacker owns a profile of the leakage according to the values of some intermediate variables, which is a strong limitation.

Recently a new kind of SCA, called *Mutual Information Analysis* (MIA), has been proposed in [4]. It uses the *Mutual Information* as distinguisher. It is an interesting alternative to the aforementioned attacks since some assumptions about the adversary can be relaxed. In particular it does not require a linear dependency between the leakage and the predicted data (as for CPA) and is actually able to exploit any kind of dependency. Moreover, this gain in generality is obtained without needing to profile the leakage as it is the case for TA.

Despite its advantages, the MIA suffers from several limitations and the preliminary work of Gierlichs *et al.* [4] poses a number of open questions. First of all, the MIA efficiency has not been clearly established and it is not clear whether (and in which contexts) it is better than the other attacks that assume the same adversary capabilities (as *e.g.* the CPA). The first attack experiments presented in [4] suggest that MIA's efficiency is strongly related to the attack context (device, algorithmic target, noise, etc.). However, at this time an in-depth analysis is missing to have a clear idea about this relationship. Secondly, the estimation of the mutual information, which itself requires the estimation of statistical distributions, is a major practical issue that has not been fully investigated in [4]. This problematic has been dealt with in Statistics and Applied Probabilities Theory (see for instance [5] for an overview). Among the existing estimation methods, it is of crucial interest to determine the one that optimizes the MIA. Only such a study will indeed allow us to form an unbiased opinion about its efficiency *versus* the one of attacks involving linear dependence based distinguishers.

2 Preliminaries on Probability and Information Theory

We use the calligraphic letters, like $\mathcal{X}$, to denote sets. The corresponding large letter X is then used to denote a random variable (r.v. for short) over $\mathcal{X}$, while the lowercase letter x - a particular element from $\mathcal{X}$. For every positive integer n, we denote by $\mathbf{X}$ a n-dimensional r.v. $(X_1, \cdots, X_n) \in \mathcal{X}^n$, while the lowercase letter $\mathbf{x}$ - a particular element from $\mathcal{X}^n$. To every discrete r.v. $\mathbf{X}$, one associates a probability mass function $\mathrm{p}_{\mathbf{X}}$ defined by $\mathrm{p}_{\mathbf{X}}(\mathbf{x}) = \mathrm{p}\left[\mathbf{X} = \mathbf{x}\right]$. If X is continuous, one associates to $\mathbf{X}$ its *probability density function* (pdf for short), denoted by $g_{\mathbf{X}}$: for every $\mathbf{x} \in \mathcal{X}^n$, we have $\mathrm{p}_{\mathbf{X}}\left[X_1 \leq x_1, \cdots, X_n \leq x_n\right] = \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_n} g_{\mathbf{X}}(t_1, \cdots, t_n) dt_1 \cdots dt_n$.

The *Gaussian distribution* is an important family of probability distributions, applicable in many fields. A r.v. $\mathbf{X}$ having such a distribution is said to be *Gaussian* and its pdf $g_{\mu,\Sigma}$ is defined for every $\mathbf{x} \in \mathcal{X}^n$ by:

$$g_{\mu,\Sigma}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^{\top}\Sigma^{-1}(\mathbf{x}-\mu)\right) \ , \qquad (1)$$

where μ and Σ respectively denote the *mean* and the *covariance matrix* of $\mathbf{X}$.

In this paper, we will study r.v. whose pdf is a finite linear combination of Gaussian pdfs. Such a pdf, which is called a *Gaussian mixture* (GM for short), is denoted by g_θ and it satisfies for every $\mathbf{x} \in \mathcal{X}^n$:

$$g_\theta(\mathbf{x}) = \sum_{t=1}^{T} a_t g_{\mu_t, \Sigma_t}(\mathbf{x}) \ \ , \tag{2}$$

where $\theta = ((a_t, \mu_t, \Sigma_t))_{1 \le t \le T}$ is a $3T$-dimensional vector containing the so-called *mixing probabilities* a_t's (that satisfy $\sum_t a_t = 1$), as well as the means μ_t and the covariance matrices Σ_t of the T Gaussian pdfs in the mixture.

The *entropy* $\mathrm{H}(\mathbf{X})$ of a discrete n-dimensional r.v. $\mathbf{X}$ aims at measuring the amount of information provided by an observation of $\mathbf{X}$. It is defined by $\mathrm{H}(\mathbf{X}) = -\sum_{\mathbf{x} \in \mathcal{X}^n} \mathrm{p}_{\mathbf{X}}(\mathbf{x}) \log_2(\mathrm{p}_{\mathbf{X}}(\mathbf{x}))$. The *differential entropy* extends the notion of entropy to continuous n-dimensional r.v. Contrary to the entropy, the differential entropy can be negative. It is defined by:

$$\mathrm{H}(\mathbf{X}) = -\int_{\mathbf{x} \in \mathcal{X}^n} g_{\mathbf{X}}(\mathbf{x}) \log_2(g_{\mathbf{X}}(\mathbf{x})) d\mathbf{x} \ \ . \tag{3}$$

If $\mathbf{X}$ is a n-dimensional Gaussian r.v. with pdf $g_{\mu,\Sigma}$, then its entropy satisfies:

$$\mathrm{H}(\mathbf{X}) = \frac{1}{2} \log((2\pi e)^n |\Sigma|) \ \ . \tag{4}$$

In the general case when $\mathbf{X}$ has a GM pdf mixing more than one Gaussian pdf, there is no analytical expression for its differential entropy. However, upper and lower bounds can be derived. We recall hereafter the lower bound.

Proposition 1. [6] *Let $\mathbf{X} \in \mathcal{X}^n$ be a Gaussian mixture whose pdf g_θ is such that $\theta = ((a_i, \mu_i, \Sigma_i))_{i=1,\cdots,T}$. Then, its differential entropy satisfies:*

$$\frac{1}{2} \log\left((2\pi e)^n \prod_{t=1}^{T} |\Sigma_t|^{a_t} \right) \le H(\mathbf{X}) \ \ . \tag{5}$$

To quantify the amount of information that a second r.v. $\mathbf{Y}$ reveals about $\mathbf{X}$, the notion of *mutual information* is usually involved. It is the value $\mathrm{I}(\mathbf{X}, \mathbf{Y})$ defined by $\mathrm{I}(\mathbf{X}, \mathbf{Y}) = \mathrm{H}(\mathbf{X}) - \mathrm{H}(\mathbf{X}|\mathbf{Y})$, where $\mathrm{H}(\mathbf{X}|\mathbf{Y})$ is called the *conditional entropy of* $\mathbf{X}$ *knowing* $\mathbf{Y}$. If $\mathbf{Y}$ is discrete, then it is defined by:

$$\mathrm{H}(\mathbf{X}|\mathbf{Y}) = \sum_{y \in \mathcal{Y}} \mathrm{p}_{\mathbf{Y}}(y) \mathrm{H}(\mathbf{X}|\mathbf{Y} = y) \ \ , \tag{6}$$

Thanks to the mutual information (or to the conditional entropy), we have a way to decide about the dependency of two multi-variate random variables: $\mathbf{X}$ and $\mathbf{Y}$ are *independent* iff $I(\mathbf{X}, \mathbf{Y})$ equals 0 or equivalently iff $\mathrm{H}(\mathbf{X}|\mathbf{Y}) = \mathrm{H}(\mathbf{X})$.

3 Brief Overview of Side Channel Attacks

Any intermediate variable which is a function $f(X, k^\star)$ of a plaintext X and a guessable secret key $k^\star$ is *sensitive* and its manipulation can be targeted by an SCA. For every key-candidate $k \in \mathcal{K}$, we denote by f_k the function $x \mapsto f(x, k)$ and by $L(k^\star)$ the *leakage variable* that models the leakage produced by the manipulation/computation of $f_{k^\star}(X)$ by the device. The leakage variable can be expressed as:

$$L(k^\star) = \varphi \circ f_{k^\star}(X) + B \ , \tag{7}$$

where φ denotes a deterministic function and B denotes an independent noise.

In (7), the definition of f only depends on the algorithm that is implemented and it is known to the attacker (it can for instance be a S-box function). On the opposite, φ only depends on the device and its exact definition is usually unknown to the attacker who will estimate it according to the device specifications and/or to a leakage profiling phase. Actually, the SCAs essentially differ in the degree of knowledge on φ and B that is required for the attack to succeed.

In a DPA, the attacker only needs to know that the mean of the r.v. $\varphi \circ f_{k^\star}(X)$ depends on a given bit of $f_{k^\star}(X)$. Based on this assumption, each key candidate k is involved to split the measurements into two sets and the candidates are discriminated by computing differences of means between those sets. This essentially amounts to process a Boolean correlation.

In a CPA, the attacker must know a function $\hat{\varphi}$ that is a good linear approximation of φ (*i.e.* such that $\hat{\varphi}$ and φ are linearly correlated). Usually, he chooses the Hamming weight function for $\hat{\varphi}$. Based on this assumption, key candidates k are discriminated by testing the linear correlation between $\hat{\varphi} \circ f_k(x_i)$ and $L(k^\star)$ for a sample of plaintexts $(x_i)_i$. This attack can be more efficient than the single-bit DPA. However, its success highly depends on the correctness of the linear approximation of φ by $\hat{\varphi}$.

In a TA, the attacker must know a good approximation of the pdf of the leakage $L(k)$ for every possible key k. It amounts for the attacker to have a good approximation of φ and of the standard deviation of the noise B (or its covariance matrix in a multivariate model). Wrong key hypotheses are discriminated in a maximum likelihood attack (see [3]). To pre-compute the pdf's of all the variables $L(k)$, the attacker needs to have an open access to a copy of the device under attack. This is a strong constraint since it is often very difficult to have such an open copy in practice.

As noticed in [4,5], MIA attacks are an alternative to the approaches above. They consist in estimating the mutual information $\mathrm{I}(L(k^\star), \hat{\varphi} \circ f_k(X))$ instead of the correlation coefficient or the difference of means. In an MIA, the attacker is potentially allowed to make weaker assumptions on φ than in the CPA. Indeed, he does not need a good linear approximation of φ but only a function $\hat{\varphi}$ s.t. the mutual information $\mathrm{I}(\hat{\varphi}, \varphi)$ is non-negligible (which may happen even if φ and $\hat{\varphi}$ are not linearly correlated). It for instance allows the attacker to choose the identity function for $\hat{\varphi}$ which is of particular interest since no knowledge about the leakage parameters is required.

The effectiveness of a key-recovery side channel attack is usually characterized by its *success rate*, namely the probability that the attack outputs the correct key as a the most likely key candidate. This notion can be extended to higher orders [7]: an attack is said to be *o-th order successful* if it classifies the correct key among the o most likely key candidates. In the following, we shall investigate the (o-th order) success rate of MIA.

Let us denote by $Z(k)$ the r.v. $\hat{\varphi} \circ f_k(X)$. Moreover, for every function F defined over $\mathcal{K}$, let us denote by argmax-o $_{k\in\mathcal{K}}$ $F(k)$ the set composed of the o key candidates k such that $F(k)$ is among the o highest values in $\{F(k); k \in \mathcal{K}\}$. An MIA succeeds at the o-th order iff the estimations $\hat{\mathrm{I}}(L(k^\star), Z(k))$ of $\mathrm{I}(L(k^\star), Z(k))$ satisfy:

$$k^\star \in \underset{k\in\mathcal{K}}{\text{argmax-}o}\ \ \hat{\mathrm{I}}(L(k^\star), Z(k)) \ . \tag{8}$$

We therefore deduce two necessary conditions for an MIA to succeed at the o-th order:

- *Theoretical.* The mutual information $(\mathrm{I}(L(k^\star), Z(k)))_{k\in\mathcal{K}}$ must satisfy:
$$k^\star \in \underset{k\in\mathcal{K}}{\text{argmax-}o}\ \ \mathrm{I}(L(k^\star), Z(k)) \ . \tag{9}$$
- *Practical.* The estimations of $(\mathrm{I}(L(k^\star), Z(k)))_{k\in\mathcal{K}}$ must be good enough to satisfy (8) while (9) is satisfied.

In the next section, we study when Relation (9) is satisfied. This will allow us to characterize (with regards to f, φ, $\hat{\varphi}$) when an MIA is theoretically possible. Then, for 3-tuples $(f, \varphi, \hat{\varphi})$ s.t. (9) is satisfied, we shall study in Sec. 6 the success probability of the MIA according to the estimation method used to compute $\hat{\mathrm{I}}$ and according to the noise variation. This will allow us to characterize when an MIA is practically feasible (*i.e.* when (8) is satisfied) and when it is more efficient than the other SCA attacks.

4 Study of the MIA in the Gaussian Model

In this section we focus on first order MIA and, in a second time, we extend our analysis to the higher order case *i.e.* when the target implementation is protected by masking [8]. Our analyses are done under the three following assumptions which are realistic in a side channel analysis context and make the formalization easier.

Assumption 1 (Uniformity). *The plaintext X has a uniform distribution over $\mathbb{F}_2^n$.*

Assumption 2 (Balancedness). *For every $k \in \mathcal{K}$, the (n, m)-function f_k : $x \mapsto f_k(x)$ is s.t. $\#\{x \in \mathbb{F}_2^n; y = f_k(x)\}$ equals 2^{n-m} for every $y \in \mathbb{F}_2^m$.*

Remark 1. This assumption states that the algorithmic functions targeted by the SCA are balanced which is usually the case in a cryptographic context.

Assumption 3 (Gaussian Noise). *The noise B in the leakage (see (7)) has a Gaussian distribution with zero mean and standard deviation σ.*

Remark 2. This assumption is realistic and is therefore often done in the literature (see for instance [8,9,7]). Practical attacks and pdf estimations presented in Sec. 6 provide us with an experimental validation of this assumption.

For clarity reasons, in the next sections we shall denote by L (resp. by Z) the random variable $L(k^\star)$ (resp. $Z(k)$) when there is no ambiguity.

4.1 First Order MIA

The mutual information $\mathrm{I}(L, Z(k))$ equals $\mathrm{H}(L) - \mathrm{H}(L, Z(k))$. Since $\mathrm{H}(L)$ does not depend on the key prediction, $\mathrm{I}(L|Z(k))$ reaches one of its o highest values when k ranges over $\mathcal{K}$ iff the conditional entropy $\mathrm{H}(L|Z(k))$ reaches one of its o smallest values. One deduces that an MIA is theoretically possible iff the 3-tuple $(f, \varphi, \hat{\varphi})$ is s.t.:

$$k^\star \in \underset{k\in\mathcal{K}}{\text{argmin-}o}\ \ \mathrm{H}(L(k^\star)|Z(k)) \ , \tag{10}$$

where argmin-o is defined analogously to argmax-o .

The starting point of our analysis is that studying the MIA effectiveness is equivalent to investigating the minimality of $\mathrm{H}(L|Z(k))$ over $\mathcal{K}$. As a consequence of (6), we have $\mathrm{H}(L|Z(k)) = \sum_{z\in\mathrm{Im}(\hat{\varphi})} \mathrm{p}_Z(z)\mathrm{H}(L|Z(k) = z)$. From (3), one deduces:

$$\mathrm{H}(L|Z(k)) = -\sum_{z\in\mathrm{Im}(\hat{\varphi})} \mathrm{p}_Z(z) \int_\ell g_{L|Z=z}(\ell) \log g_{L|Z=z}(\ell) d\ell \ . \tag{11}$$

To reveal the relationship between $\mathrm{H}(L|Z(k))$ and the key-prediction k, the expression of the pdf $g_{L|Z=z}$ in (11) needs to be developed. Let us denote by $E_k(z)$ the set $[\hat{\varphi} \circ f_k]^{-1}(z)$. Since X has a uniform distribution over $\mathbb{F}_2^n$, for every $\ell \in \mathcal{L}$ and every $z \in \mathrm{Im}(\hat{\varphi} \circ f_k)$ we have:

$$g_{L|Z=z}(\ell) = \frac{1}{\#E_k(z)} \sum_{x\in E_k(z)} g_{\varphi\circ f_{k^\star}(x),\sigma}(\ell) \ . \tag{12}$$

The next proposition directly follows.

Proposition 2. *If X is a r.v. with uniform distribution, then for every pair $(k^\star, k) \in \mathcal{K}^2$ and every $z \in \mathcal{Z}$ the pdf of the r.v. $(L(k^\star) \mid Z(k) = z)$ is a GM g_θ whose parameter θ satisfies $\theta = \left((a_{z,t}, t, \sigma^2)\right)_{t\in Im(\varphi)}$, with $a_{z,t} = \mathrm{p}\left[\varphi \circ f_{k^\star}(X) = t \mid \hat{\varphi} \circ f_k(X) = z\right]$.*

In Proposition 2, the key hypothesis k only plays a part in the definition of the weights $a_{z,t}$ of the GM. In other terms, $g_{L|Z(k)=z}$ is always composed of the same Gaussian pdfs and the key hypothesis k only impacts the way how the Gaussian pdfs are mixed. To go further in the study of the relationship between k and $\mathrm{H}(L(k^\star)|Z(k)=z)$, let us introduce the following diagram where z is an element of $\mathrm{Im}(\hat{\varphi})$, where F', F and T are image sets:

$$\begin{tikzcd} z \arrow[r, "\hat{\varphi}^{-1}"] & F' \arrow[r, "f_k^{-1}"] & E_k(z) \arrow[r, "f_{k^\star}"] & F \arrow[r, "\varphi"] & T \end{tikzcd} ,$$

Based on the diagram above, we can make the two following observations:

- If the set T is reduced to a singleton set $\{t_1\}$ (*i.e.* if $\hat{\varphi} \circ f_k$ is constant equal to t_1 on $E_k(z)$), then all the probabilities $a_{z,t}$ s.t. $t \neq t_1$ are null and a_{z,t_1} equals 1. In this case, one deduces from Proposition 2 that the distribution of $(L(k^\star)|Z(k)=z)$ is Gaussian and, due to (4), its conditional entropy satisfies
$$\mathrm{H}(L(k^\star)|Z(k)=z) = \frac{1}{2}\log(2\pi e\sigma^2) \ .$$
- If $\#T > 1$ (*i.e.* if $\#\varphi \circ f_{k^\star}(E_k(z)) > 1$), then there exist at least two probabilities a_{z,t_1} and a_{z,t_2} which are non-null and the distribution of $(L(k^\star)|Z(k) = z)$ is a GM (not Gaussian). Due to (5), its entropy satisfies:
$$\mathrm{H}(L(k^\star)|Z(k)=z) \geq \frac{1}{2}\log(2\pi e\sigma^2) \ .$$

When φ is constant on F' (*e.g.* when $\hat{\varphi} = \varphi$ or $\hat{\varphi} = \mathrm{Id}$), the two observations above provide us with a discriminant property. If $k^\star = k$, then we have $F = F'$ and thus T is a singleton and $\mathrm{H}(L|Z=z)$ equals $\frac{1}{2}\log(2\pi e\sigma^2)$. Otherwise, if $k \neq k^\star$, then $f_{k^\star} \circ f_k$ is likely to behave as a random function[1]. In this case, F is most of the time different from F' and T is therefore likely to have more than one element[2]. This implies that $\#\varphi \circ f_{k^\star}(E_k(z))$ is strictly greater than 1 and thus that $\mathrm{H}(L|Z=z)$ is greater than or equal to $\frac{1}{2}\log(2\pi e\sigma^2)$. Eventually, we get the following proposition in which we exhibit a tight lower bound for the differential entropy $\mathrm{H}(L(k^\star)|Z(k))$.

Proposition 3. *For every $(k^\star, k) \in \mathcal{K}^2$, the conditional entropy of the r.v. $(L(k^\star)|Z(k))$ satisfies:*

$$\frac{1}{2}\log\left(2\pi e\sigma^2\right) \leq H((L(k^\star)|Z(k)) \ . \tag{13}$$

If $\varphi \circ f_{k^\star}$ is constant on $E_k(z)$ for every $z \in \mathcal{Z}$, then the lower bound is tight.

Proof. Relation (13) is a straightforward consequence of (6) and of Propositions 1 and 2. The tightness is a direct consequence of (4) and Proposition 2. ◇

[1] This property, sometimes called *wrong key assumption* [10], is often assumed to be true in a cryptographic context, due to the specific properties of the primitive f.

[2] As detailed later, this is only true if $\hat{\varphi} \circ f_k$ is non-injective.

Remark 3. Intuitively, the entropy $\mathrm{H}(Y)$ is a measure of the diversity or randomness of Y. It is therefore reasonable to think that the more components in the GM pdf of $(L(k^\star)|Z(k))$, the greater its entropy. Relation (13) provides a first validation of this intuition. The entropy is minimal when the pdf is a Gaussian one (*i.e.* when the GM has only one component). In our experiments (partially reported in Sec. 6), we noticed that the entropy of a GM whose components have the same variance, increases with the number of components.

Corollary 1. *If $\hat{\varphi} \circ f_k$ is injective, then $H(L(k^\star)|Z(k))$ equals $\frac{1}{2}\log(2\pi e\sigma^2)$.*

Proof. If $\hat{\varphi} \circ f_k$ is injective, then $E_k(z)$ is a singleton and $\varphi \circ f_{k^\star}$ is thus constant on $E_k(z)$.

If the functions $\hat{\varphi} \circ f_k$'s are all injective, then Corollary 1 implies that the MIA cannot succeed at any order. Indeed, in this case the entropy $\mathrm{H}(L(k^\star)|Z(k))$ stays unchanged when k ranges over $\mathcal{K}$ and thus, $k^\star$ does not satisfy (10). As a consequence, when the f_k's are injective (which is for instance the case when f_k consists in a key addition followed by the AES S-box), then the attacker has to choose $\hat{\varphi}$ to be non-injective (*e.g.* the Hamming weight function). It must be noticed that this is a necessary but not sufficient condition since the function $\hat{\varphi}$ must also be s.t. $\mathrm{I}(\hat{\varphi}, \varphi)$ is non-negligible (otherwise the MIA would clearly failed). In this case, the attacker must have a certain knowledge about the leakage function φ in order to define an appropriate function $\hat{\varphi}$ and hence, the MIA does no longer benefit from one of its main advantages. This drawback can be overcome by exclusively targeting intermediate variables s.t. the f_k's are not injective (in AES, the attacker can for instance target the bitwise addition between two S-box outputs during the *MixColumns* operation).

4.2 Generalization to the Higher Order Case

In this section, we extend the analysis of MIA to higher orders and we assume that the implementation protected by masking. The sensitive variable $f_{k^\star}(X)$ is now masked with $d-1$ independent random variables M_1, ..., M_{d-1} which are uniformly distributed over $\mathrm{Im}(f)$.

The masked data $f_{k^\star}(X) \oplus M_1 \oplus \cdots \oplus M_{d-1}$ and the different masks M_j's are processed at different times. The leakage about $f_{k^\star}(X) \oplus M_1 \oplus \cdots \oplus M_{d-1}$ is denoted by L_0 and the leakages about the M_j's are denoted by L_1, ..., L_{d-1}. Under Assumption 3, the L_j's satisfy:

$$L_j = \begin{cases} \varphi[f_{k^\star}(X) \oplus \bigoplus_{t=1}^{d-1} M_t] + B_0 & \text{if } j = 0, \\ \varphi_j(M_j) + B_j & \text{if } j \neq 0, \end{cases} \tag{14}$$

where the B_j's are independent Gaussian noises with mean 0 and standard deviations σ_j, and where φ, φ_1, $\cdots$, φ_{d-1} are d device dependent functions that are *a priori* unknown to the attacker. The vector $(L_0, \cdots, L_{d-1})$ is denoted by $\mathbf{L}$. The vector of masks $(M_1, \cdots, M_{d-1})$ is denoted by $\mathbf{M}$. We denote by $\Phi_{k^\star}(X, \mathbf{M})$ the vector $(\varphi(f_{k^\star}(X) \oplus \bigoplus_{t=1}^{d-1} M_t), \varphi_1(M_1), \cdots, \varphi_{d-1}(M_{d-1}))$.

To simplify our analysis, we assume that the attacker knows the manipulation times exactly and is therefore able to get a sample for the r.v. $\mathbf{L}$. Under this assumption and for the same reasons as in the univariate case, the higher order MIA essentially consists in looking for the key candidate k which minimizes an estimation of the conditional entropy $\mathrm{H}(\mathbf{L}|Z(k))$. Due to (6), this entropy equals $\sum_{z\in\mathrm{Im}(\hat{\varphi})} \mathrm{p}_{Z(k)}(z)\mathrm{H}(\mathbf{L}|Z(k)=z)$. Since Z equals $\hat{\varphi}\circ f_k(X)$, the probabilities $\mathrm{p}_{Z(k)}(z)$ in this sum can be exactly computed by the attacker. Once this computation has been performed, estimating $\mathrm{H}(\mathbf{L}|Z(k))$ amounts to estimate the entropies $\mathrm{H}(\mathbf{L}|Z(k)=z)$ for all the hypotheses k. These entropies are estimated as for the first order case (see (11)), but the pdfs $g_{\mathbf{L}|Z(k)=z}$ are multivariate. More precisely, after denoting by Σ the matrix $(\mathrm{Cov}\,[B_i,B_j])_{i,j}$, we get:

$$g_{\mathbf{L}|Z(k)=z}(\ell) = \frac{1}{\#E_k(z)(\#Im(f))^{d-1}} \sum_{\substack{x\in E_k(z)\\ \mathbf{m}\in Im(f)^{d-1}}} g_{\Phi_{k^\star}(x,\mathbf{m}),\Sigma}(\ell)\ . \tag{15}$$

In a similar way than in Sec. 4, the next proposition directly follows.

Proposition 4. *If X is a r.v. with uniform distribution, then for every pair $(k^\star,k)\in\mathcal{K}^2$ and every $z\in\mathcal{Z}$ the pdf of the r.v. $(\mathbf{L}(k^\star)\mid Z(k)=z)$ is a GM g_θ whose parameter θ satisfies $\theta=((a_{z,\mathbf{t}},\mathbf{t},\Sigma))_{\mathbf{t}}$, with $\Sigma=(Cov\,[B_i,B_j])_{i,j}$ and $a_{z,\mathbf{t}}=\mathrm{p}[\Phi_{k^\star}(X,\mathbf{M})=\mathbf{t}\mid\hat{\varphi}\circ f_k(X)=z]$.*

We deduce from Propositions 1 and 4 the following result.

Proposition 5. *If X is a r.v. with uniform distribution over $\mathcal{X}$, then for every $(k^\star,k)\in\mathcal{K}^2$, the entropy of the r.v. $(\mathbf{L}(k^\star)|(Z(k),\mathbf{M}))$ satisfies:*

$$\frac{1}{2}\log\left((2\pi e)^d|\Sigma|\right) \le H(\mathbf{L}(k^\star)|(Z(k),\mathbf{M}))\ . \tag{16}$$

If $\varphi\circ f_{k^\star}$ is constant on $E_k(z)$ for every $z\in Im(Z)$, then the bound is tight.

We cannot deduce from the proposition above a wrong-key discriminator as we did in the univariate case. Indeed, to compute the entropy in (16) the attacker must know the mask values, which is impossible in our context. However, if the 3-tuple $(f,\varphi,\hat{\varphi})$ satisfies the condition of Proposition 5, then it can be checked that for every z the number of components in the multi-variate GM pdf of $(L(k^\star)|Z(k)=z)$ reaches its minimum for $k=k^\star$. As discussed in Remark 3, this implies that the entropy of $\mathbf{L}(k^\star)|Z(k)$ is likely to be minimum for $k=k^\star$. The simulations and experiments presented in Sec. 6 provides us with an experimental validation of this fact.

In the next sections, we assume that an MIA is theoretically possible. Namely, we assume that $k^\star$ belongs to argmin-o $_k$ $\mathrm{H}(\mathbf{L}(k^\star)|Z(k))$ for a given order o. At first, we study the success probability of an MIA according to the method used to estimate $\mathrm{H}(\mathbf{L}(k^\star)|Z(k))$ and the noise variation. Secondly, we compare the efficiency of an MIA with the one of the CPA in different contexts.

5 Conditional Entropy Estimation

Let $\mathbf{L}$ be a d-dimensional r.v. defined over $\mathcal{L}^d$ (*i.e.* $\mathbf{L}$ is composed of d different instantaneous leakage measurements) and let k be a key-candidate. We assume that the attacker has a sample of N leakage-message pairs $(\mathbf{l}_i, x_i) \in \mathcal{L}^d \times \mathcal{X}$ corresponding to a key $k^\star$, and that he wants to compute $\mathrm{H}(\mathbf{L}|Z(k))$ to discriminate key-candidates k. Due to (6), estimating $\mathrm{H}(\mathbf{L}|Z(k))$ from the sample $((\mathbf{l}_i, x_i))_i$ essentially amounts to estimate the entropy $\mathrm{H}(\mathbf{L}|Z(k) = z)$ for every $z \in \mathcal{Z}$. For such a purpose, a first step is to compute estimations $\hat{g}_{\mathbf{L}|Z=z}$ of the $g_{\mathbf{L}|Z=z}$'s. Then, depending on the estimation method that has been applied, the $\mathrm{H}(\mathbf{L}|Z(k) = z)$'s are either directly computable (Histogram method) or must still be estimated (Kernel and Parametric methods). In the following we present three estimation methods and we discuss their pertinency in our context.

5.1 Histogram Method

Description. We choose d *bin widths* h_0, ..., h_{d-1} (one for each coordinate of the leakage vectors) and we partition the leakage space $\mathcal{L}^d$ into regions $(\mathcal{R}_\alpha)_\alpha$ with equal volume $\upsilon = \prod_j h_j$. Let k be a key-candidate and let z be an element of $\mathcal{Z}$. We denote by $\mathcal{S}_z$ the sub-sample $\left(\mathbf{l}_i; x_i \in [\varphi \circ f_k]^{-1}(z)\right)_i \subseteq (\mathbf{l}_i)_i$ and by $\ell_{i,j}$ the jth coordinate of $\mathbf{l}_i$. To estimate the pdf $g_{\mathbf{L}|Z=z}$, we first compute the density vector D_z whose coordinates are defined by:

$$D_z(\alpha) = \frac{\#(\mathcal{S}_z \cap \mathcal{R}_\alpha)}{\#\mathcal{S}_z} \ , \tag{17}$$

where $\mathcal{S}_z \cap \mathcal{R}_\alpha$ denotes the sample of all the $\mathbf{l}_i$'s in $\mathcal{S}_z$ that belong to $\mathcal{R}_\alpha$.

The estimation $\hat{g}_{L|Z=z}$ is then defined for every $\mathbf{l} \in \mathcal{L}^d$ by $\hat{g}_{\mathbf{L}|Z=z}(\mathbf{l}) = \frac{D_z(i_\mathbf{l})}{\upsilon}$, where $i_\mathbf{l}$ is the index of the region $\mathcal{R}_{i_\mathbf{l}}$ that contains $\mathbf{l}$. Integrating the pdf estimation according to formula (3) gives the following estimation for the conditional entropy: $\hat{\mathrm{H}}(\mathbf{L}|Z = z) = -\sum_\alpha D_z(\alpha) \log(D_z(\alpha)/\upsilon)$. We eventually get:

$$\hat{\mathrm{H}}(\mathbf{L}|Z) = -\sum_{z \in \mathcal{Z}} \mathrm{p}_Z(z) \sum_\alpha D_z(\alpha) \log\left((\frac{D_z(\alpha)}{\upsilon}\right) \ . \tag{18}$$

The optimal choice of the bin widths h_j is an issue in Statistical Theory. Actually, there are several rules that aim at providing *ad hoc* formulae for computing the h_j's based on the nature of the samples (see for instance [11,12]). In our simulations, we chose to follow the Scott Rule. Namely, if $\hat{\sigma}_j$ denotes the estimated standard deviation of the sample $(\ell_{i,j})_i$ of size N_j, then h_j satisfies $h_j = 3.49 \times \hat{\sigma}_j \times N_j^{-\frac{1}{3}}$ (notice that in our context all the N_j's are equal to N).

Simulations. In order to illustrate the Histogram method in the context of an MIA attack, we generated 10000 leakage measurements in the Gaussian model (7) for φ being the Hamming weight function, for f being the first DES S-box parameterized with the key $k^\star = 11$ and for $\sigma = 0.1$. Since the DES S-box is

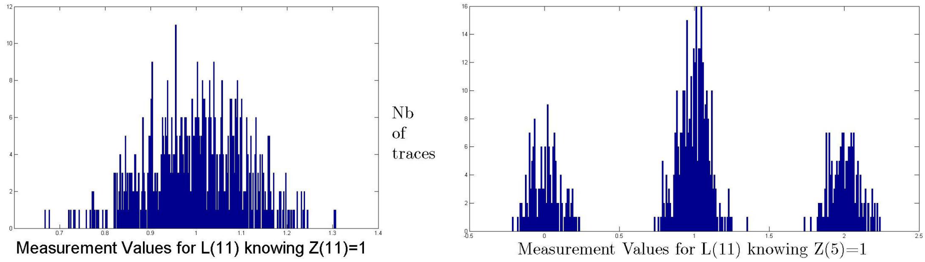


Fig. 1. Histogram Method in the First Order Case

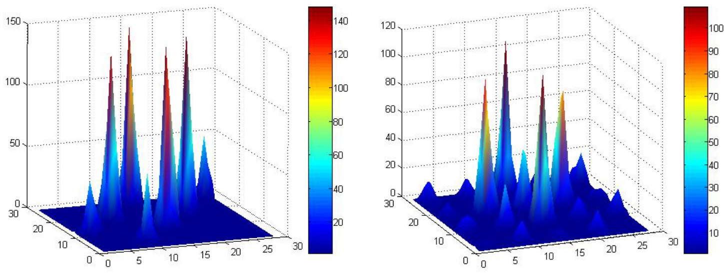


Fig. 2. Histogram Method in the Second Order Case

non-injective, we chose the identity function for $\hat{\varphi}$. Fig. 1 plots the estimations of the pdf $g_{L|Z=1}$ when $k = 11$ and when $k = 5$ (for a number of bins equal to 285). As expected (Proposition 2 and Corollary 1), a Gaussian pdf seems to be estimated when $k = 11$ (good key prediction), whereas a mixture of three Gaussian distributions seems to be estimated when $k = 5$ (wrong key prediction). For the experimentation described in the left-hand figure we obtained $\hat{\mathrm{H}}(L(11)|Z(11) = 1) = -1.31$ (due to (4) we have $\mathrm{H}(L(11)|Z(11) = 1) = -1.27$) and we got $\hat{\mathrm{H}}(L(11)|Z(5) = 1) = -0.0345$ for the one in the right-hand side. Moreover, we validated that the estimated conditional entropy is minimum for the good key hypothesis.

In order to illustrate the Histogram method in the context of a 2nd order MIA attack, we generated 10000 pairs of leakage measurements in the higher order Gaussian model (14) with $d = 2$, with φ and φ_1 being the Hamming Weight function, with f being the first DES S-box parametric with the key $k^{\star} = 11$ and with $\sigma_0 = \sigma_1 = 0.1$. Fig. 2 plots the estimations of the pdf $g_{\mathbf{L}|Z=1}$ when $k = 11$ and when $k = 5$. As expected, the mixture of Gaussian distributions for $k = 11$ have less components than for $k = 5$. For the experimentation in the left-hand figure we obtained $\hat{\mathrm{H}}(\mathbf{L}(11)|Z(11) = 1) = 0.22$ (and $\hat{\mathrm{H}}(\mathbf{L}(11)|Z(11)) = 0.14$), whereas we got 1.12 for $\hat{\mathrm{H}}(\mathbf{L}(11)|Z(5) = 1)$ (and 1.15 for $\hat{\mathrm{H}}(\mathbf{L}(11)|Z(5))$). Here again, the estimated conditional entropy was minimum for the good key hypothesis.

5.2 Kernel Density Method

Description. Although the Histogram method can be made to be asymptotically consistent, other methods can be used that converge at faster rates. For instance, rather than grouping observations together in bins, the so-called *Kernel density estimator* (or *Parzen window* method) can be thought to place small "bumps" at each observation, determined by the Kernel function (see for instance [13]). The estimator consists of a "sum of bumps" and is clearly smoother as a result than the Histogram method.

The Kernel density estimation $\hat{g}_{\mathbf{L}|Z=z}$ based on the sample $\mathcal{S}_z$ is defined for every $\mathbf{l} = (\ell_0, ..., \ell_{d-1}) \in \mathcal{L}^d$ by:

$$\hat{g}_{L|Z=z}(\mathbf{l}) = \frac{1}{\#\mathcal{S}_z} \sum_{\mathbf{l}_i=(\ell_{i,0},\ldots,\ell_{i,d-1})\in\mathcal{S}_z} \frac{1}{\upsilon} \times \prod_{j=0}^{d-1} \mathbf{K}\left(\frac{\ell_j - \ell_{i,j}}{h_j}\right) \enspace ,$$

where $\mathbf{K}$ is a *Kernel function* chosen among the classical ones (see for instance [14]), where the h_i's are *Kernel bandwidths* and where υ equals $\prod_j h_j$. As recalled in [15], the following Parzen-windows entropy estimation of $\mathrm{H}(\mathbf{L}|Z = z)$ is sound when the sample size is large enough:

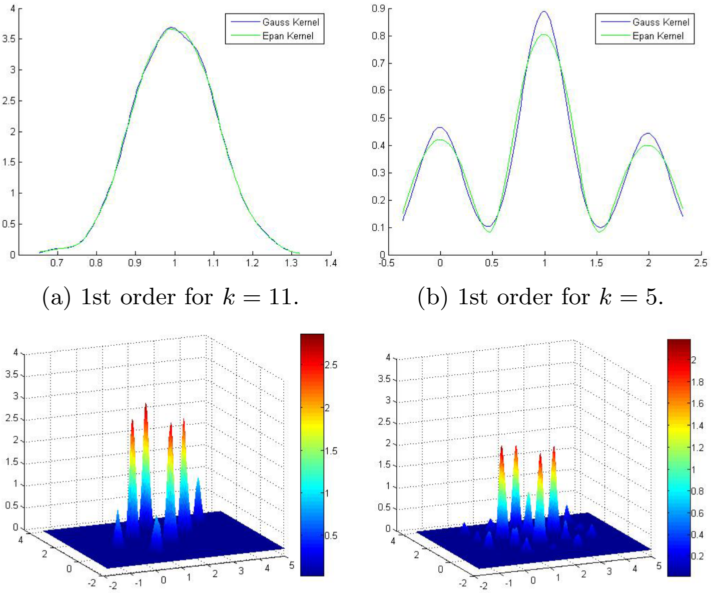
$$\hat{\mathrm{H}}(\mathbf{L}|Z = z) = -\frac{1}{\#\mathcal{S}_z} \sum_{\mathbf{l}_i\in\mathcal{S}_z} \log\left(\frac{1}{\#\mathcal{S}_z} \sum_{\mathbf{l}_r\in\mathcal{S}_z} \frac{1}{\upsilon} \times \prod_{j=0}^{d-1} \mathbf{K}\left(\frac{\ell_{i,j} - \ell_{r,j}}{h_j}\right)\right) \enspace ,$$

In our attack simulations, we chose the Kernel function to be the Epanechnikov one defined for every u by $\mathbf{K}(u) = \frac{3}{4}(1-u^2)$ if $|u| \leq 1$ and by $\mathbf{K}(u) = 0$ otherwise (another usual choice is the Gaussian Kernel [14]). Our choice was motivated not only by the fact that this Kernel function has a simple form, but also by the fact that its efficiency is asymptotically optimal among all the Kernels [16]. Let $\hat{\sigma}_j$ denotes the estimated standard deviation of the sample $(\ell_{i,j})_i$ of size N_j. To select the Kernel bandwidth h_j, we followed the *normal scale rule* [13]. Namely, we chose the h_j's s.t. $h_j = 1.06 \times \hat{\sigma}_j \times N_j^{-\frac{1}{5}}$.

Simulations. In order to illustrate the effectiveness of the Kernel method, we applied it for the same simulated traces used for our 1st and 2nd order Histogram experiments (Fig. 1 and Fig. 2). We present our results in Fig. 3(a–b) for the first order and in Fig. 3(c–d) for the second order.

As expected, the pdf estimated in Fig. 3(a) when $k = 11$ seems to be a Gaussian one, whereas the pdf estimated when $k = 5$ seem to be a mixture of three Gaussian distributions. Moreover, the estimations are smoother than in the case of the Histogram Method and there is no noticeable differences between the estimation with Gaussian Kernel and the estimation with the Epanechnikov one. For the experimentation described in the left-hand figure we obtained $\mathrm{H}(L(11)|Z(11) = 1) = -0.88$ and we got 0.54 for $\mathrm{H}(L(11)|Z(5) = 1)$ (right-hand side).

As expected, in Fig. 3(c) the mixture of Gaussian distributions for $k = 11$ have less components than for $k = 5$. For the experimentation in the left-hand figure



(a) 1st order for $k = 11$.

(b) 1st order for $k = 5$.

(c) 2nd order for $k = 11$.

(d) 2nd order for $k = 5$.

Fig. 3. Kernel Method

we obtained $\mathrm{H}(\mathbf{L}(11)|Z(11)) = 0.17$, whereas we got 0.52 for $\mathrm{H}(\mathbf{L}(11)|Z(5))$. Moreover, we validated that the conditional entropy $\mathrm{H}(\mathbf{L}(11)|Z(k))$ is minimum for $k = k^\star = 11$.

5.3 Parametric Estimation

Description. Under Assumption 3, (12) shows that $g_{L|Z=z}$ is a GM g_θ whose parameter θ satisfies:

$$\theta = \left(\frac{1}{\#E_k(z)}, \varphi \circ f(x, k^\star), \sigma^2\right)_{x \in E_k(z)} . \tag{19}$$

Based on this relation, an alternative to the methods presented above is to compute an estimation $\hat{\theta}$ of the parameter θ so that we get $\hat{g}_{L|Z=z} = g_{\hat{\theta}}$ and thus:

$$\hat{\mathrm{H}}(\mathbf{L}|Z = z) = -\int_{\mathbf{l} \in \mathcal{L}^d} g_{\hat{\theta}}(\mathbf{l}) \log_2 g_{\hat{\theta}}(\mathbf{l}) d\mathbf{l} .$$

For every x, the mean value $\varphi \circ f(x, k^\star)$ in (19) can be estimated by $\bar{\mathbf{l}}_x = \frac{1}{\#\{i; x_i = x\}} \sum_{i; x_i = x} \mathbf{l}_i$ and the noise variance σ^2 by $\hat{\sigma}^2 = \sum_i \left(\mathbf{l}_i - \bar{\mathbf{l}}_{x_i}\right)^2$. On the whole, this provides us with the following estimation $\hat{\theta}$ of θ:

$$\hat{\theta} = \left(\frac{1}{\#E_k(z)}, \bar{\mathbf{l}}_x, \hat{\sigma}^2\right)_{x \in E_k(z)} .$$

For Higher Order MIA, (15) can be rewritten:

$$g_\theta = \frac{1}{\#E_k(z)} \sum_{x \in E_k(z)} g_{\theta_x} \ , \tag{20}$$

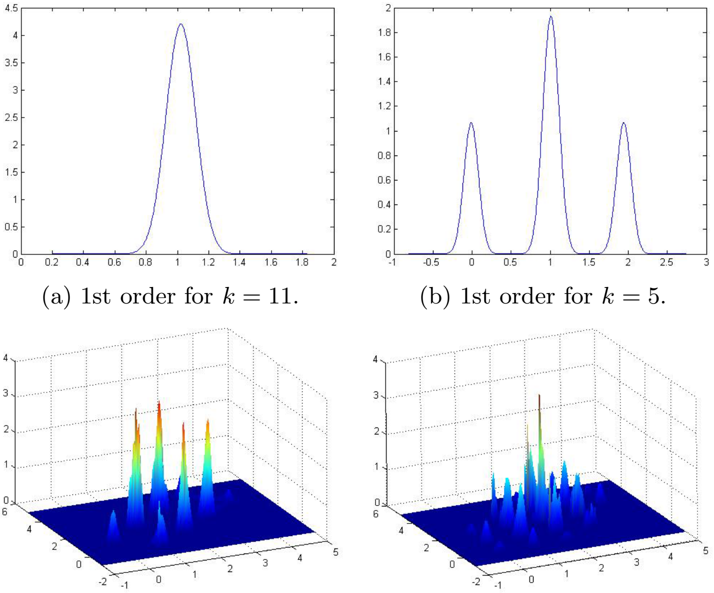
where g_{θ_x} denotes the GM pdf of the r.v. $(\mathbf{L}|X = x)$ whose parameter satisfies:

$$\theta_x = \left(\frac{1}{(\#Im(f))^{d-1}}, \Phi_k(x, \mathbf{m}), \Sigma \right)_{\mathbf{m} \in Im(f)^{d-1}} \ .$$

The mean values $\Phi_k(x, \mathbf{m})$ of the different components cannot be directly estimated as in the first order case since the values taken by the masks $\mathbf{m}$ for the different leakage observations $\mathbf{l}_i$ are not assumed to be known. To deal with this issue, a solution is to involve GM estimation methods such as the *Expectation Maximization Algorithm.* By applying it on the sample $(\mathbf{l}_i \ ; \ x_i = x)_i$ we get an estimation $\hat{\theta}_x$ of θ_x for every $x \in \mathcal{X}$. Then, according to (20), we obtain:

$$\hat{\mathrm{H}}(\mathbf{L} \mid Z = z) = -\sum_x \int_{\mathbf{l} \in \mathcal{L}^d} g_{\hat{\theta}_x}(\mathbf{l}) \log g_{\hat{\theta}_x}(\mathbf{l}) d\mathbf{l} \ .$$

Remark 4. As an advantage of the Parametric estimation method, the mean values $\mathbf{l}_x$'s (resp. the estimated parameters $\hat{\theta}_x$'s) are only computed once for every x and are then used to compute $\hat{\mathrm{H}}(\mathbf{L}|Z(k) = z)$ for every pair (k, z).



(a) 1st order for $k = 11$.

(b) 1st order for $k = 5$.

(c) 2nd order for $k = 11$.

(d) 2nd order for $k = 5$.

Fig. 4. Parametric Estimation

Simulations. As for the previous estimation methods, we applied the Parametric estimation to the same simulated traces. The resulting estimated pdfs $(\hat{g}_{\mathbf{L}(11)|Z(k)=1})_{k=11,5}$ are plotted in Fig. 4(a–b) for the first order and in Fig. 4(c–d) for the second order.

The results are similar to those of the previous estimation methods. For the first order case, we distinguish a mixture of three Gaussian distributions for the wrong key hypothesis while a single Gaussian pdf is observed for the correct one. For the second order case, the GM obtained for the wrong key hypothesis contains more components than the one for the correct key hypothesis. Once again, the estimated entropy is lower for the correct key hypothesis than for the wrong one. For instance, the entropies of the plotted pdfs equal -0.94 (correct hyp.) and 0.13 (wrong hyp.) for the first order case and 0.24 (correct hyp.) and 0.60 (wrong hyp.) for the second order case.

6 Experimental Results

6.1 First Order Attack Simulations

To compare the efficiency of the MIA with respect to the estimation method, we simulated leakage measurements in the Gaussian model (7) with φ being the Hamming weight function and f being the first DES S-box (we therefore have $n = 6$ and $m = 4$). For various noise standard deviations σ and for the estimation methods described in previous sections, we estimated the number of messages required to have an attack first order success rate greater than or equal to 90% (this success rate being computed for 1000 attacks). Moreover, we included the first Order CPA in our tests to determine whether and when an MIA is more efficient than a CPA[3]. Each attack was performed with $\hat{\varphi}$ being the identity function in order to test the context in which the attacker has no knowledge about the leakage model. Moreover, each attack was also performed with $\hat{\varphi}$ being the Hamming weight function in order to test the context where the attacker has a good knowledge of the leakage model. The results are given in Table 1 where MIA$_H$, MIA$_K$ and MIA$_P$ respectively stand for the Histogram, the Kernel and the Parametric MIA.

It can be checked in Table 1 that the CPA is always better than the MIA when $\hat{\varphi} = \text{HW}$. This is not an astonishing result in our model, since the deterministic part of the leakage corresponds to the Hamming weight of the target variable. More surprisingly, this stays true when $\hat{\varphi}$ is chosen to be the identity function. This can be explained by the strong linear dependency between the identity function and the Hamming weight function over $\mathbb{F}_2^4 = \{0, \ldots, 15\}$. Eventually, both results suggest that the CPA is more suitable than the MIA for attacking a device leaking first order information in a model close to the Hamming weight model with Gaussian noise. When looking at the different MIAs, we can notice that MIA$_P$ becomes much more efficient than MIA$_H$ and MIA$_K$ when the noise standard deviation increases.

[3] Attacks have been performed for measurements numbers ranging over 50 different values from 30 to 10^6.

Table 1. Attack on the first DES S-box – Number of measurements required to achieve a success rate of 90% according to the noise standard deviation σ

Attack \ σ	0.5	1	2	5	10	15	20	50	100
CPA, $\hat{\varphi} = \mathrm{Id}$	30	30	100	1000	3000	7000	15000	70000	260000
MIA$_H$ (Hist), $\hat{\varphi} = \mathrm{Id}$	80	160	600	4000	20000	50000	95000	850000	10^6+
MIA$_K$ (Kernel), $\hat{\varphi} = \mathrm{Id}$	70	140	500	3000	15000	35000	60000	500000	10^6+
MIA$_P$ (Param.), $\hat{\varphi} = \mathrm{Id}$	60	100	300	2000	5000	15000	20000	150000	500000
CPA, $\hat{\varphi} = \mathrm{HW}$	30	30	70	400	2000	4000	7000	45000	170000
MIA$_H$ (Hist), $\hat{\varphi} = \mathrm{HW}$	40	70	300	1500	7000	20000	40000	320000	10^6+
MIA$_K$ (Kernel), $\hat{\varphi} = \mathrm{HW}$	30	60	190	1500	5500	15000	25000	190000	900000
MIA$_P$ (Param.), $\hat{\varphi} = \mathrm{HW}$	70	70	150	1000	3000	7000	15000	65000	300000

6.2 Second Order Attack Simulations

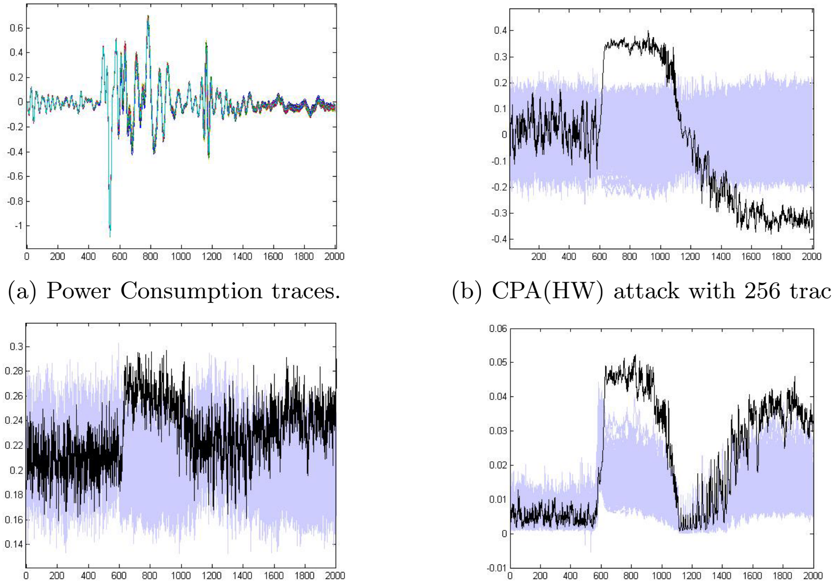
In a CPA, the attacker computes Pearson correlation coefficients which is a function of two univariate samples. Thus, when the CPA is applied against dth order masking (see ([14])) a multivariate function must be defined to combine the different leakage signals (corresponding to the masked data and the masks) [9]. This signal processing induces an information loss which strongly impacts the Higher order CPA efficiency when the noise is increasing. Because an Higher Order MIA can operate on multivariate samples, it does not suffer from the aforementioned drawback. We can therefore expect the MIA to become more efficient than the CPA when it is performed against masking. To check this intuition, we simulated power consumption measurements such as in ([14]) with $d = 2$, with $\varphi = \varphi_1 = \mathrm{Id}$, with $\sigma_1 = \sigma_2 = \sigma$ and with f being the first DES S-box. For various noise standard deviations σ and for the estimation methods described in previous sections, we estimated the number of measurements required to have an attack success rate greater than or equal to 90% (this success rate being computed over 100 attacks). In the following table, we compare second order MIA with Histogram estimation method (2O-MIA$_H$) with second order CPA (2O-CPA) for two different *combining function*[4].

Table 2. Second Order Attack on DES S-box – Number of measurements required to achieve a success rate of 90% according to the noise standard deviation σ

Attack σ	0.5	1	2	5	7	10
2O-CPA ($\hat{\varphi} = \mathrm{HW}$, abs. diff. combining))	300	800	5000	200000	10^6+	10^6+
2O-CPA ($\hat{\varphi} = \mathrm{HW}$, norm. product combining)	300	400	3000	70000	300000	10^6+
2O-MIA$_H$ ($\hat{\varphi} = \mathrm{Id}$)	7000	7000	8000	15000	30000	55000

The results presented in Table 2 corroborate our intuition: when the noise standard deviation crosses the threshold 4, second order MIA attacks become

[4] In our simulations we performed 2O-CPAs involving either the absolute difference or the normalized product combining [9].



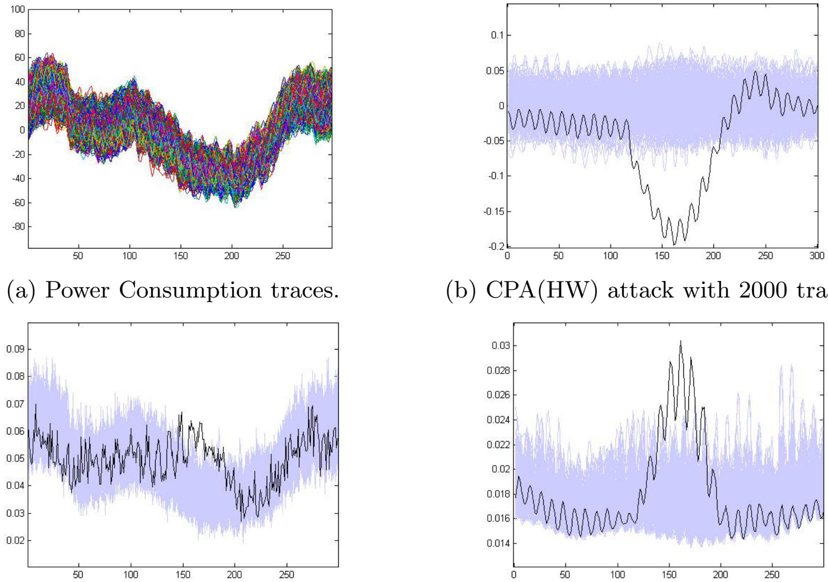
(a) Power Consumption traces.

(b) CPA(HW) attack with 256 traces.

(c) MIA (Hist) attack with 1024 traces.

(d) MIA (param.) attack with 1024 traces.

Fig. 5. Practical Attacks on a Hardware AES Implementation



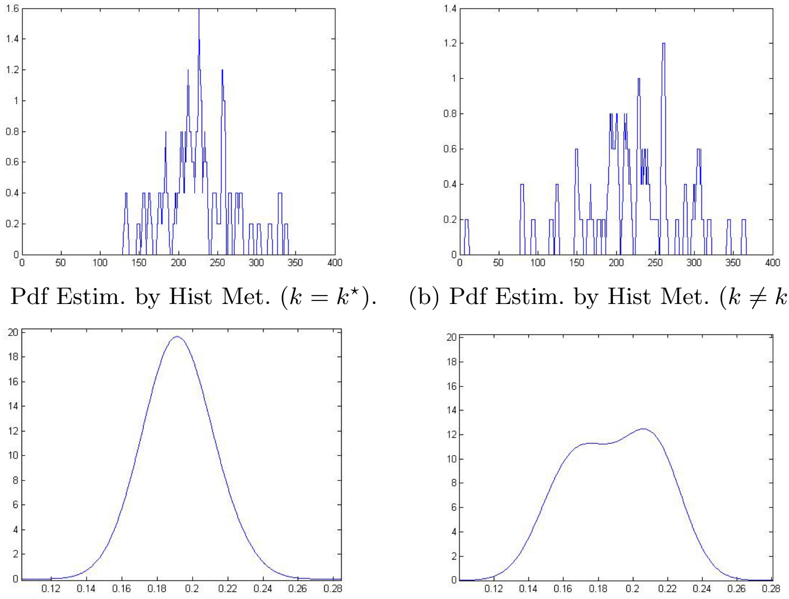
(a) Power Consumption traces.

(b) CPA(HW) attack with 2000 traces.

(c) MIA (Hist) attack with 2000 traces.

(d) MIA (Param.) attack with 2000 traces.

Fig. 6. Practical Attacks on a Software AES Implementation



(a) Pdf Estim. by Hist Met. ($k = k^\star$). (b) Pdf Estim. by Hist Met. ($k \neq k^\star$).

(c) Param. pdf Estim. ($k = k^\star$). (d) Param. pdf Estim. ($k \neq k^\star$).

Fig. 7. Pdf estimations on power measurements

much more efficient than second order CPA even for leakage measurements simulated in the Gaussian Model with φ = HW which is favorable to CPA-like attacks.

6.3 Practical Attacks

To test the MIA in a real-life context, we performed it against two AES S-box implementations that use a lookup-table (*i.e.* f_k corresponds to the AES S-box). The first one is a hardware implementation on the chip SecMat V3/2 (see [17] for details about the chip and the circuit's layout). The corresponding power consumption measurements are plotted in Fig. 5(a) over the time. It can be noticed that they are not very noisy. The second one is a software implementation running on a 8-bit architecture smart card. As it can be seen in Fig. 6(a), the signal is much more noisy in this case.

For both set of traces, we performed the CPA and the MIA attacks with the Histogram estimation method and the Parametric estimation method (see Sec. 5). For all of these attacks the prediction function $\hat{\varphi}$ was chosen to be the Hamming weight function (since $\hat{\varphi} \circ f_k$ must be non-injective – see Corollary 1 –). The obtained correlation and mutual information curves are plotted in Fig. 5(b–d) and Fig. 6(b–d) over the time. For each attack the curve corresponding to the correct (resp. wrong) key hypothesis is drawn in black (resp. gray).

In both cases, the attacks succeed with a few number of traces. It can be noticed that the MIA with a Parametric estimation is more discriminating than the

MIA with the Histogram estimation. This confirms the simulations performed in Sec. 6.1. However, even when the Parametric estimation method is involved, the CPA is always more discriminating than the MIA. Those results suggest that for the attacked devices the power consumption has in fact a high linear dependency with the Hamming weight of the manipulated data. This implies in particular that the Hamming weight Model is sound in this context and that looking for non-linear dependencies is not useful.

To corroborate that the leakage measured in Fig. 5 and 6 are close to the one simulated in Sec. 5, we plotted in Fig. 7 the estimation of the pdf $g_{\mathbf{L}(0)|Z(k)=1}$ when $k = 0 = k^\star$ and $k = 5 \neq k^\star$ for the hardware implementation. We could verify that actually the conditional pdfs that are estimated look like GM pdfs (a Gaussian pdf when $k^\star$ is correctly guessed and a mixture of two pdfs when it is not).

7 Conclusion

This paper extends the works published in [4] and [5] to expose the theoretical foundations behind this attack and it generalizes it to higher orders. This analysis clarifies assets and limitations of the MIA. In particular, it shows that the MIA is less efficient than the CPA when the deterministic part of the leakage is a linear function of the prediction made by the attacker. This implies that the CPA must be preferred to the MIA when the targeted device leaks a linear function of the Hamming weight of the manipulated data. This paper also argues that the way to estimate the mutual information has an impact on the attack efficiency. A parametric estimation method has been introduced which renders the MIA efficiency close to the one of the CPA when the noise is increasing. When masking is used to protect the implementation, an extension of the MIA has been proposed which is, for our simulations, much more efficient than classical higher order CPA. It actually seems that this is the context in which the MIA offers an efficient alternative to correlation-based attacks.

References

1. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: [18], pp. 388–397
2. 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)
3. 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. 13–29. Springer, Heidelberg (2003)
4. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008)
5. Aumonier, S.: Generalized Correlation Power Analysis. In: Proceedings of the Ecrypt Workshop Tools For Cryptanalysis 2007 (2007)

6. Carreira-Perpinan, M.: Mode-finding for mixtures of Gaussian distributions Carreira-Perpinan. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(11), 1318–1323 (2000)
7. Standaert, F.X., Malkin, T.G., Yung, M.: A Formal Practice-Oriented Model For The Analysis of Side-Channel Attacks. Cryptology ePrint Archive, Report 2006/139 (2006)
8. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: [18], pp. 398–412
9. Prouff, E., Rivain, M., Bévan, R.: Statistical Analysis of Second Order Differential Power Analysis. IEEE Transactions on Computers (to appear, 2009)
10. Canteaut, A., Trabbia, M.: Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 573–588. Springer, Heidelberg (2000)
11. Turlach, B.A.: Bandwidth selection in kernel density estimation: A review. In: CORE and Institut de Statistique, pp. 23–493 (1993)
12. Wand, M.P.: Data-based choice of histogram bin width. The American Statistician 51, 59–64 (1997)
13. Silverman, B.: Density Estimation for Statistics and Data Analysis. Chapman and Hall, Boca Raton (1986)
14. Wasserman, L.: All of Statistics: A Concise Course in Statistical Inference. Springer Texts in Statistics (2005)
15. Beirlant, J., Dudewicz, E.J., Györfi, L., Meulen, E.C.: Nonparametric entropy estimation: An overview. International Journal of the Mathematical Statistics Sciences 6, 17–39 (1997)
16. Gray, A.G., Moore, A.W.: Nonparametric density estimation: Toward computational tractability. In: Proceedings of the Third SIAM International Conference on Data Mining. SIAM, Philadelphia (2003)
17. Guilley, S., Sauvage, L., Hoogvorst, P., Pacalet, R., Bertoni, G.M., Chaudhuri, S.: Security evaluation of wddl and seclib countermeasures against power attacks. IEEE Transactions on Computers 57(11), 1482–1497 (2008)
18. Wiener, M. (ed.): CRYPTO 1999. LNCS, vol. 1666. Springer, Heidelberg (1999)

Attacking ECDSA-Enabled RFID Devices

Michael Hutter, Marcel Medwed, Daniel Hein, and Johannes Wolkerstorfer

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
{mhutter,mmedwed,dhein,jwolkers}@iaik.tugraz.at

Abstract. The elliptic curve digital signature algorithm (ECDSA) is used in many devices to provide authentication. In the last few years, more and more ECDSA implementations have been proposed that allow the integration into resource-constrained devices like RFID tags. Their resistance against power-analysis attacks has not been scrutinized so far. In this article, we provide first results of power-analysis attacks on an RFID device that implements ECDSA. To this end, we designed and implemented a passive RFID-tag prototype. The core element of the prototype is a low-power ECDSA implementation realized on 180 nm CMOS technology. We performed power and electromagnetic attacks on that platform and describe an attack that successfully reveals the private-key during signature generation. Our experiments confirm that ECDSA-enabled RFID tags are susceptible to these attacks. Hence, it is important that they implement countermeasures which prevent the forging of digital signatures.

Keywords: Radio-Frequency Identification, RFID, Side-Channel Analysis, ECDSA, Elliptic Curve Cryptography, Implementation Security.

1 Introduction

Radio Frequency Identification (RFID) is an emerging technology that is becoming more and more important in our daily life. There already exist billions of RFID devices and their integration into existing applications seems almost inevitable. Due to the widespread use of this wireless technology, security issues have become a primary concern in the past few years. Especially public-key enabled RFID devices have gained importance because they allow an easier and more secure key management than symmetric solutions. This article focuses on the security of RFID devices that can generate elliptic curve digital signatures.

RFID devices consist of a tiny microchip that is connected to an antenna. These so-called tags are typically powered passively by a reader via an electromagnetic field. This field is also used for the communication between the tags and the reader. Typical security applications of RFID are access-control systems, cashless payment, and the electronic passport. These applications need the devices to make use of cryptographic algorithms to provide required services such as authentication, confidentiality, integrity, and non-repudiation. The cryptographic algorithms have to be light-weight in terms of power and area to

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 519–534, 2009.

cope with the limited resources in large-scale RFID applications. Due to these constraints, most tags that are now available typically rely on proprietary algorithms or symmetric primitives that have been proven to be suitable for RFID. Large effort has been made in this field to enable resource-constrained implementations of algorithms such as AES [5], DESL [18], PRESENT [3], and HIGHT [9]. Although symmetric algorithms can provide many of the required security assurances, the advantages of facilitated key management offered by asymmetric algorithms would be very desirable for RFID systems. However, they are much more complex to implement. In respect to these facts, many light-weight solutions have been proposed such as NTRU [8], XTR [19], and elliptic-curve based schemes. A typical application that uses asymmetric cryptography is the electronic passport. It comes with an embedded microchip that is used to prove the origin of the passport and to authenticate the owner. This process is referred to as *active authentication* and uses standardized algorithms such as RSA [30], DSA [24], or ECDSA [2].

Besides the growing demand for RFID devices and their widespread integration into existing security applications, there have been many articles published during the last decade that emphasize the vulnerability of such cryptographic devices against implementation attacks. Amongst the most powerful attacks are side-channel attacks that were first introduced by Kocher et al. [16] in 1996. These attacks allow the extraction of the secret key by measuring the power consumption [17], electromagnetic emanation [1,6,29], or timing information [16]. In the context of RFID, Oren and Shamir [27] have shown the first side-channel attack which allows to reveal the *kill* password of ultra-high frequency (UHF) tags, in 2006. They performed a simple power analysis (SPA) attack by observing one power trace that is reflected from the tag to the reader. This backscattered power trace changes depending on the processed data. The first differential power analysis (DPA) attack on RFID devices has been performed by Hutter et al. [10] in 2007. They analyzed hardware and software AES implementations of high frequency (HF) tag prototypes by means of power and electromagnetic analysis. All devices have been successfully attacked using less than 1 000 power traces. Plos [28] demonstrated the susceptibility of UHF tags against DPA attacks in 2008. He analyzed commercially-available RFID tags and determined data-dependent emanations at a distance of up to one meter. However, all available publications describe attacks on either kill-password extraction or symmetric primitives. So far, there exist no article that investigates DPA attacks on public-key enabled RFID tags.

In this article, we provide the first results of side-channel attacks on a public-key enabled RFID device. In order to evaluate the effectiveness of such attacks on RFID tags, we designed a prototype that is able to be powered passively by the field of a reader. The prototype includes a low-power hardware ECDSA implementation fabricated in 180 nm CMOS technology. The implementation is able to authenticate itself to a reader by generating digital signatures. Furthermore, we are the first who provide a DPA attack on a hardware implementation of ECDSA. We show how to reveal the private key during signature generation by measuring the electromagnetic emanation of the tag. In addition, we describe a useful pre-processing technique for improving side-channel attacks on RFID by applying a trace-decimation technique. All attacks have been successful and led us to the conclusion that public-key enabled RFID devices are as vulnerable as symmetric-based RFID devices. It has been shown that wireless devices are susceptible to these attacks as much as contact-based powered devices.

The article is structured as follows. Section 2 describes power-analysis attacks on passive RFID devices in general. Section 3 is dedicated to attacks on ECDSA implementations. We describe the exploitation of different information leakages of ECDSA and propose a DPA attack on the private-key operations during signature generation. Section 4 details our RFID-tag prototype. In Section 5 we describe the measurement setup and side-channel pre-processing techniques that are used in our experiments to perform power-analysis attacks. The results of our experiments are presented in Section 6. Conclusions are drawn in Section 7.

2 Power Analysis of Passive RFID Devices

Side-channel analysis of passive RFID devices is a challenging task due to several reasons. In this section, we give an overview on various issues regarding the acquisition and analysis of side-channel information that are exploited from passive RFID tags.

Passive tags differ from conventional contact-based devices in several ways. First, passive tags only possess two antenna connections. Indeed, there are no dedicated power-supply pins available where a resistor can be placed in series to measure the consumed power. An alternative way of side-channel extraction is the sensing of electromagnetic emanation. The current flow within the microchip of the RFID tag produces an electromagnetic field. This field contains different signals such as the square-wave clock or signals that are caused by data-dependent processing. These signals can be sensed by magnetic near-field probes that are placed directly on the surface of the chip [1]. However, while such attacks will succeed for many contact-based devices, this may not be the case for passive RFID tags. Passive tags have been designed for low-power operation and consume only a few micro Watts of power. Special measurement equipment is therefore necessary to separate and amplify the weak side-channel signals that are emitted from the tags.

In RFID environments, we are actually concerned with another source of electromagnetic emanation. There is not only the weak emanation of the tags but also the emanation of the reader device. This reader field is typically between 40 dB and 80 dB higher than the signals emitted by the tags. As a result, interesting signal emissions of the tags may be unintentionally overwhelmed by the occurring interferences of the reader. The data-acquisition resolution of the measurement equipment is thus inevitably reduced since the weak signals of the tag are superimposed onto the much higher reader field. In addition to the lower acquisition resolution, this reader field is not synchronized with the measurement equipment which causes power-trace misalignments in both the time and the amplitude dimension. The reader is a high noise source and therefore makes side-channel analysis difficult to perform. The main challenge of electromagnetic measurements in this context is therefore to minimize the impact of this reader signal and to overcome the resulting misalignment of measured power traces.

Another issue which is of major concern in RFID environments is the compression of side-channel traces. Passive RFID tags are powered by the electromagnetic field of a reader. Most of these tags also extract the clock signal out of this field. In order to comply with the low-power requirement, they often use a low clock frequency in the kHz range. The processing of data and especially the computation of asymmetric functions therefore takes a long time (up to several milliseconds). Side-channel attacks on public-key enabled RFID devices require compression techniques to reduce the complexity of storing and subsequent processing of millions of sample points that are acquired throughout the tag computation.

3 Power Analysis Attacks on ECDSA Implementations

ECDSA is the elliptic curve-based variant of the digital signature algorithm (DSA). The DSA has been proposed in 1991 by the National Institute of Standards and Technology (NIST). Since then, many organizations have standardized ECDSA such as ANSI [2], IEEE [11], FIPS [24], and ISO/IEC [14]. In the following, we describe ECDSA in greater detail, discuss various power-analysis attacks that have been performed on different implementations, and present a DPA attack that reveals the private key during signature calculation.

In order to generate a digital signature using ECDSA, a message m is given as an input. By using the domain parameters $D = (q, FR, S, a, b, P, n, h)$, a random number k is first chosen in the interval from 1 to n. This random number is often referred to as ephemeral key. Then, an elliptic-curve point multiplication is performed using k and the base point P. The result is converted to an integer $\bar{x}_1$ in order to compute the intermediate value r. After that, the message m is hashed using the SHA-1 algorithm [26]. The signature generation is then generated within two steps. First, the private key d is multiplied with the intermediate value r. The result is then added to the output of the hashed message $e = h(m)$. Second, the value s is calculated by inverting the ephemeral key k and multiplying it with the output of step one. The generated ECDSA signature that is returned consists of the tuple (r, s). Algorithm 1 shows the signature-generation scheme.

There exist many articles that present power-analysis attacks on elliptic curve-based algorithms. One of the first publications is due to Coron [4] in the year 1999. He showed that the scalar multiplication is highly susceptible to SPA attacks. One way to implement the scalar-multiplication is to use the double-and-add algorithm. By simply inspecting one measured power trace of such implementations, a difference in the power consumption can be observed depending on whether doubling or adding was performed. Several countermeasures have been proposed including scalar blinding techniques [4], unified point operations [15],

Algorithm 1. Signature-generation scheme using ECDSA

```
Require: Domain parameters $D = (q, FR, S, a, b, P, n, h)$, private key $d$, message $m$.
Ensure: Signature $(r, s)$
1: Select $k \in [1, n-1]$
2: Compute $[k]P = (x_1, y_1)$ and convert $x_1$ to an integer $\bar{x}_1$
3: Compute $r = \bar{x}_1 \mod n$. If $r = 0$ then go back to step 1.
4: Compute $e = H(m)$.
5: Compute $s = k^{-1}(e + dr) \pmod{n}$. If $s = 0$ then go back to step 1.
6: Return $(r, s)$
```

or the Montgomery point ladder [22]. Nevertheless, recently Medwed et al. [21] have shown attacks on implementations by using template-based SPA attacks. They have been able to successfully reveal the ephemeral key of implementations including SPA countermeasures. However, up to now there neither exist articles describing DPA attacks on ECDSA nor are there publications that reveal the private key directly instead of extracting the ephemeral key to calculate the private key afterwards.

3.1 Our Contribution and Description of the Attack

In the following, we describe a DPA attack that reveals the private key during signature generation. The target of the attack is an intermediate value that depends on the private key on the one hand and that depends on some random value on the other hand. Regarding the ECDSA scheme described in Algorithm 1, the private key d is multiplied with the output of the scalar multiplication r. The private key is static and the output of the scalar multiplication is random since the ephemeral key k is chosen randomly for each signature generation. Furthermore, r is publicly known because it is part of the signature. In the light of these facts, we are able to perform a DPA attack on intermediate values that are processed during the calculation of the multi-precision integer multiplication $d * r$.

Common hardware implementations for multi-precision multiplication are the operand scanning and the product scanning (Comba) algorithm which are depicted in Figure 1. Both algorithms multiply the words of two n-word long operands. In our case, those are r_i (the input) and d_j (the key). The resulting partial products are then added to a cumulative sum p. This results in n^2 partial products. Note that one word of the private key is processed n times during the whole integer multiplication.

What seems obvious at first glance turns out to be more complex in practice. The integer multiplication is a linear function that multiplies a constant value with a random input value. That means that shifted bit combinations of a key word have a linear impact to the multiplication result. When the key is shifted x times, the result is also shifted x times. Therefore, it is evident that in power-analysis attacks, one or more correlation peaks occur for only one key word. This is due to the fact that all bit combinations of the key word will result in

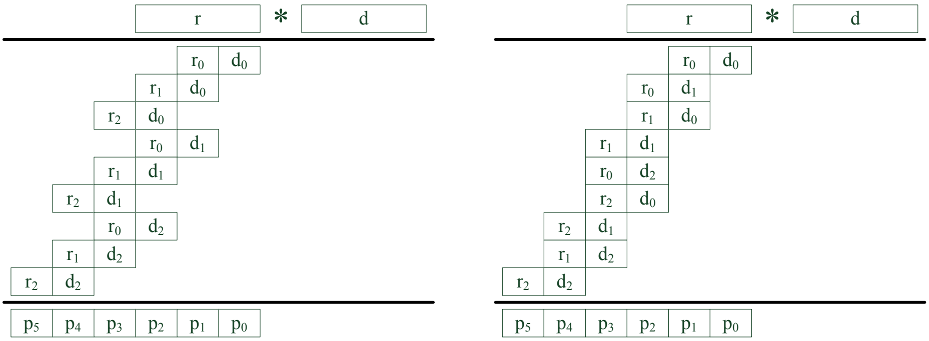


Fig. 1. Operand scanning form (left) and product scanning form (right)

the same Hamming-weight[1] value of the multiplication output. The number of possible shifts s of the key word d_i can be calculated as follows:

$$s(d_i, l) = log_2(gcd(d_i, 2^l)) + l - \lfloor log_2(d_i) + 1 \rfloor, \qquad (1)$$

where l represents the word size. Note that the maximum number of key shifts is equal to the word size, i.e. 16 for a device using 16 bit operands (in this case the key word has a Hamming weight of one and the value of the shifted key combinations are a multiple of 2^x where $x = 0..15$). Due to these facts, the decision of which hypothetical key is the correct one and which are incorrect keys seems therefore infeasible. It is clear that this makes a DPA attack much more inefficient compared to attacks on intermediate values that occur after non-linear functions such as the S-box in DES [23] or AES [25].

Our attack can be separated into two steps. In the first step, we target the output of all partial products and perform a DPA attack on that intermediate value. For each partial product, we obtain one or more promising key candidates due to the reasons described above. For a device with a 16 bit word size, for example, we obtain up to 16 promising key candidates. Hence, we get up to 16 key candidates for each private-key word d_i. In the second step, we target the output of the final multiplication product p. Each word of this product depends on one or more different key words. Thus, we can use the information obtained from the first step and use all obtained key candidates d_i to perform an attack on the final product words p_i. After revealing the key candidates for d_0 and d_1, for example, we can attack the second product word p_1 to obtain the correct key word d_0. Incorrect key hypotheses will show low correlation peaks so that they can be eliminated from the correct key hypothesis that causes a higher correlation. By following this way, a DPA attack on each of these product words p_i will yield all private-key words d_i successively.

[1] The Hamming weight power model is often used in practice and is further used in order to describe the attack.

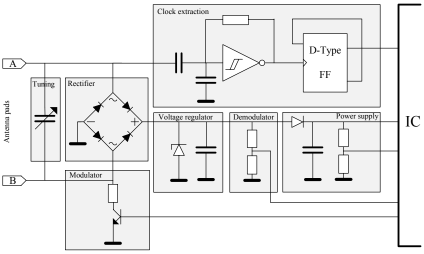


Fig. 2. Schematic of the analog front-end of our passively powered RFID-tag prototype

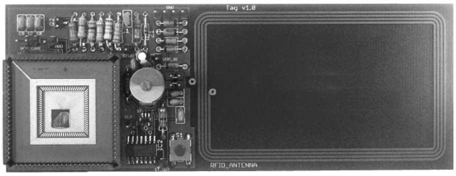


Fig. 3. A passively powered RFID-tag prototype that is capable of generating digital signatures using ECDSA

4 A Passive ECDSA-Enabled RFID-Tag Prototype

In this section, we present the design and implementation of the passively powered RFID-tag prototype that has been used throughout our experiments. The prototype consists of an antenna, an analog front-end, and a low-power digital controller. The antenna has four windings and has been designed according to ISO 7816 [12]. The antenna is connected to an analog front-end that transforms the received analog signals of the reader to the digital world of the digital controller. The controller includes a digital RFID front-end and a low-power hardware implementation of ECDSA. In the following, we describe the analog front-end and the digital controller in a more detail.

4.1 The Analog Front-End

The analog front-end is composed of the seven parts shown in the schematic view in Figure 2. In the first stage, the antenna is connected to a matching circuit which tunes the antenna to the 13.56 MHz carrier frequency of the reader. After that, a bridge rectifier has been assembled using low-voltage drop schottky diodes. The rectified signal is then smoothed and fed into a slow envelope detector to provide a stable power supply for the digital controller.

In order to comply with many commercial RFID tags, we designed a clock extraction circuit that is able to regenerate a system clock out of the 13.56 MHz reader signal. For this, a relaxation oscillator has been implemented using an inverting Schmitt trigger, one resistor, and a capacitor which produce a stable 13.56 MHz square-wave clock. The clock is then divided by two using a d-type flip-flop to provide a 6.78 MHz clock frequency that is needed for the controller.

For receiving and sending of data, both a modulation and a demodulation circuit have been integrated. For data modulation, a resistor is used that can be connected and disconnected to the antenna by the controller. After switching the resistor, a significant amount of additional power is drawn out of the reader field. This so-called load modulation is then detected and demodulated by the reader.

4.2 The Digital ECDSA-Enabled RFID Controller

The digital controller is an elliptic-curve point multiplication device with an ISO 15693 [13] compatible digital RFID front-end. It is capable of computing the multiplication of a scalar value with a point on the NIST standardized elliptic curve B-163 [24]. The B-163 curve is defined on the binary extension field $\mathbb{F}_{2^{163}}$. The controller was fabricated by the UMC L180 GII 1P/6M 1.8V/3.3V CMOS process. The controller has a total area of 15 630 Gate Equivalents (GE) while an overhead of 654 GE is incurred by components for production testing. The digital RFID front-end requires 1 726 GE and the ECC core 13 250 GE. This includes 1 346 GE for a memory slot to enable the separate setting of the ephemeral key k.

The controller must be operated at a fixed frequency of 6.78 MHz. This is half of the carrier frequency. Internally, this frequency supplies two different clock domains. One of them is used for the RFID interface and has a frequency of 106 kHz. The other one clocks the ECC core at 847.5 kHz. The whole chip has an estimated power consumption of about 176 µW.

5 The Measurement Setup

The measurement setup is composed of several parts. We used a PC, an RFID reader, the RFID-tag prototype, a digital sampling oscilloscope (DSO), a differential probe, and a near-field measurement probe. The PC controls the overall measurement process. It is connected to the DSO and the RFID reader. An 8-bit oscilloscope is used that offers an acquisition bandwidth of up to 1 GHz. As a reference measurement, an active differential probe has been connected in parallel to a 1 Ω resistor that is placed in series to the *VDD* core power supply. For electromagnetic measurements, we used a tiny magnetic near-field probe that allows the sensing of signals only up to a few millimeters. This already reduces the noisy reader signal in an early stage of the acquisition process. The sensed signals are then amplified by a 30 dB pre-amplifier before they are sampled by the oscilloscope. The sampling rate has been set to 1 GS/s for all measurements. Figure 4 shows the RFID measurement setup involving our tag prototype that lies on the antenna of a reader.

We have used a standard RFID reader that supports mandatory ISO 15693 commands such as *Inventory* or *Select*. It is also able to send custom commands that are needed to start the ECDSA signature generation. We defined three custom commands. The first command (0xE0) performs a hardware reset and loads initial data (like the base point) from Read Only Memory (ROM) into the internal Static Random Access Memory (SRAM) of the tag controller. The second command (0xE1) starts the scalar multiplication and the third command (0xE2) evaluates the signing equation given in Algorithm 1.

The communication flow between the reader and our tag prototype is shown in Figure 6. First, a reset command (0xE0) is sent to the tag. The tag responds with its unique ID (UID) number. Instead of calculating the scalar multiplication in each power trace acquisition, we pre-calculated the value r and loaded it into

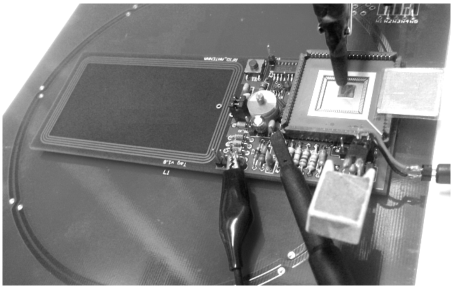


Fig. 4. RFID measurement setup involving our tag prototype lying on the reader antenna

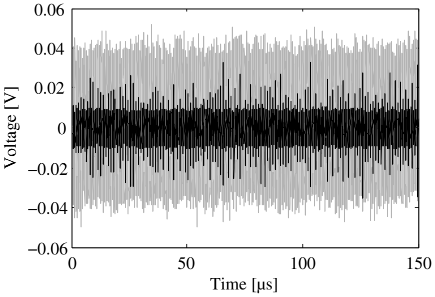


Fig. 5. Power trace (black) and (30-times magnified) EM trace (gray) during the calculation of the private-key multiplication

the SRAM before starting the power acquisition of the integer multiplication. This is done by sending the mandatory Write-Single-Block (WSB) command (0x21) of ISO 15693. After that, the reader sends the ECDSASign command (0xE2) to start the calculation of the digital signature (r, s). As a trigger signal, the oscilloscope was set to listen on the End-of-Frame (EOF) sequence of the Write-Single-Block command.

The RFID controller inverts the ephemeral key k in about 11.8 ms. The private-key multiplication needs about 150 µs, the hash-value addition and the final multiplication need around 600 µs. In our setup, the power consumption as well as the electromagnetic emanation of our device were acquired simultaneously throughout the private-key multiplication. Figure 5 shows one measured power trace (drawn in black) and a 30-times magnified electromagnetic trace (drawn in gray).

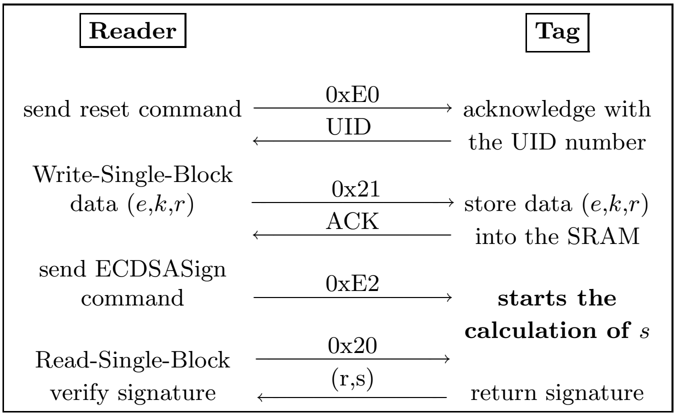


Fig. 6. RF communication between the reader and the tag

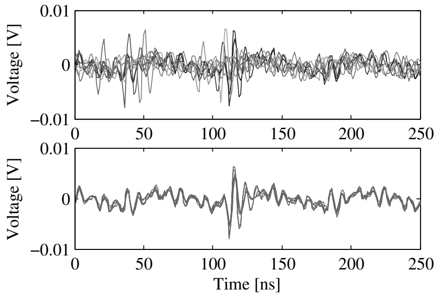


Fig. 7. Misaligned traces (upper plot) and aligned traces (lower plot) of electromagnetic measurements

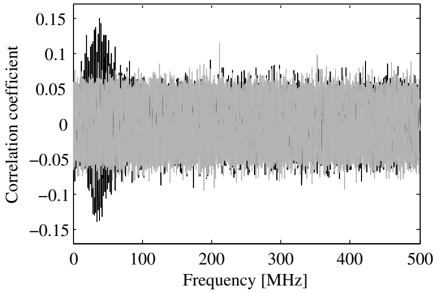


Fig. 8. Correct (black) and incorrect (gray) correlation traces of the frequency-based DPA attack using 2 000 power traces

5.1 Pre-processing RFID Power Traces

As we already stated in Section 2, measurements in RFID environments are very noisy due to the high energy signal of the reader device. We therefore applied two pre-processing techniques: trace alignment and trace decimation.

First, we aligned all measured traces in both horizontal and vertical orientation. We determined a specific trace pattern which was used to align the remaining traces using the least-mean-square (LMS) algorithm. The misaligned traces are shown in the upper plot of Figure 7. The lower plot demonstrates the traces after alignment. Our experiments have shown that without alignment or even poor alignment, successful attacks become largely infeasible due to the high noise of the measurement setup.

Second, we applied a trace-decimation technique that has been proven to be very useful throughout our experiments. Decimation is a technique typically used in signal processing. It performs two actions: filtering and re-sampling. First, the measured power traces are applied to an appropriate low-pass filter. This filter attenuates all frequency signals above a certain cut-off frequency. Second, it re-samples the smoothed traces at a lower rate. Decimation has therefore two major advantages. On the one hand, misalignment are compensated due to the averaging of filtering. On the other hand, all measured traces become shorted in their length. Both properties are a major concern for successful attacks in RFID environments as stated in Section 2.

In order to apply the decimation technique to our power traces, we determined a proper cut-off frequency. This frequency has to be chosen in a way so that signals are eliminated that do not carry data-dependent information. Agrawal et al. [1] have shown that there exist data-dependent information in the lower frequency spectrum. The higher the frequency, the weaker will be the signals that carry interesting information. Due to this fact, we have performed a frequency-based DPA attack that was first introduced by Gebotys [7]. All measured power traces are transformed into the frequency domain using the Fast Fourier Transformation (FFT). Instead of correlating the sample points in the time domain, the sample points are correlated in the frequency domain. As a target of the

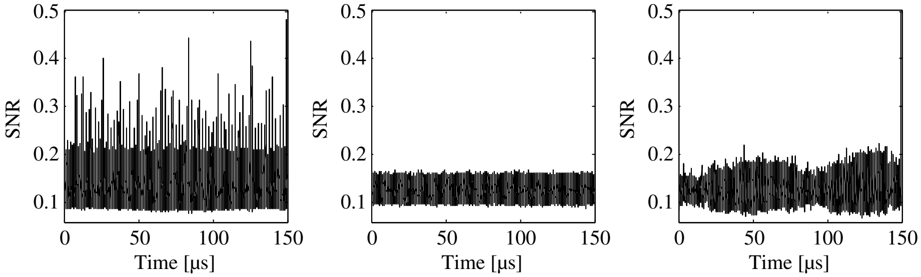


Fig. 9. SNR of the power traces (left), SNR of the EM traces without pre-processing (middle), and SNR of the pre-processed EM traces (right)

attack, we have chosen the same intermediate value used in the attack described in Section 3. Figure 8 shows the result of the attack using 2 000 power traces. The correct key hypothesis is drawn in black and the incorrect key hypotheses are drawn in gray. It can be clearly seen that there is a high correlation below 50 MHz. Above this frequency, no significant correlation can be discerned. On this account, we applied an 8th-order Chebyshev (Type 1) low-pass filter with a cutoff frequency at 50 MHz and down-sampled the traces accordingly. All traces have been decimated from 300 000 sampling points to only 32 500 samples.

5.2 Device Characterization and Pre-processing Evaluation

Next, we characterize our tag prototype concerning side-channel information leakage. First, the noise of the measurement setup is characterized. Second, the data-dependent signal that is leaked by the device is determined. After that, we calculate the signal-to-noise (SNR) ratio of the power and the electromagnetic measurements and compare them. Furthermore, we evaluate the pre-processing techniques by comparing measurements with and without trace alignment and trace decimation.

The noise of the measurement setup has been characterized by calculating the mean of all traces that were captured by processing constant data. This avoids data-dependent power variations and allows the determination of the measurement noise. Data-dependent signals, in contrast, have been characterized within two steps: First, the mean of those traces that process the same input data is calculated. Second, the variance of these mean traces is calculated. The SNR can now be calculated by dividing the variance of the obtained mean-signal trace from the variance of the calculated noise trace [20]. For the SNR calculation, 2 000 traces have been used.

Figure 9 shows the result of the characterization and performance evaluation. The left plot in the figure shows the SNR of the power traces. A maximum SNR of 0.45 has been obtained. In the middle plot of the figure, the result of the EM traces is given which have not been pre-processed. It can be seen that the signal components are below the noise floor. With this number of traces, an attack would fail due to the low SNR. The right plot of the figure shows the SNR

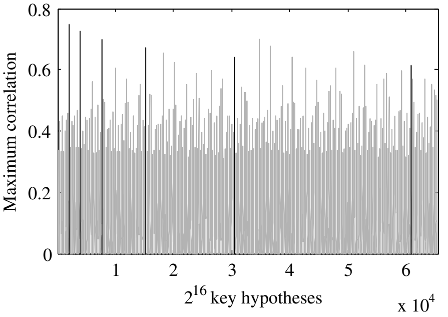


Fig. 10. Maximum correlation coefficient of all 2^{16} key hypotheses for the first private-key word d_0 using 2 000 power traces

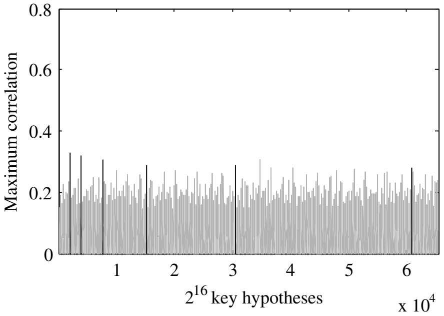


Fig. 11. Maximum correlation coefficient of all 2^{16} key hypotheses for the first private-key word d_0 using 2 000 EM traces

of the pre-processed EM traces. Due to the pre-processing, the SNR could be significantly increased to a maximum of 0.22. The signal components are much weaker as compared to the power traces but an attack will still succeed as shown in the next section.

6 Results

This section presents results of power and electromagnetic (EM) analysis attacks on our ECDSA-enabled RFID-tag prototype. First, we perform a reference attack using a contact-based power analysis. The power consumption of the RFID-tag prototype is measured over a resistor that has been placed in series to the integrated RFID controller and the analog front-end. Second, we perform a contact-less attack using EM analysis by using a magnetic near-field probe. In both scenarios, the tag was powered passively by the field.

The first attack targets the first partial product of the multi-precision multiplication unit of our RFID controller. The controller implements a 16-bit Comba-multiplication unit so that we have to test 2^{16} key hypotheses that are multiplied with the known intermediate value r. The target has been the 32-bit output of the multiplication. However, our experiments have shown that our device does not leak all bits of this 32-bit output with same amount. Hence, we have modeled the power consumption by weighting the Hamming weight of the higher 16 bits and the lower 16 bits differently to obtain the highest correlation.

Figure 10 and Figure 11 show the result of the power and EM attack. For both attacks, 2 000 traces have been used. The x-axis represents all possible key hypotheses and the y-axis represents the maximum absolute correlation of each resulting correlation trace. It is clearly discernable that the results obtained from the EM traces reach only half the correlation value as they have been obtained from the power traces. The reason for this is the lower SNR of the EM measurement calculated in the previous section. Furthermore, it can be observed that the highest peak has been obtained for the key word 1901 and reached a correlation coefficient of 0.75 for the power traces and 0.33 for the EM

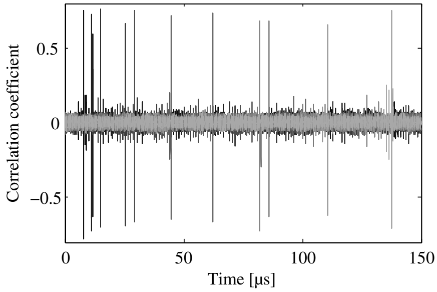


Fig. 12. Correlation traces of all partial products $r_i * d_0$ using 2 000 power traces

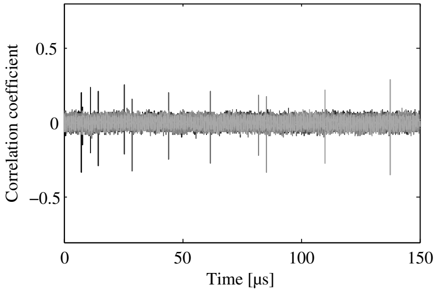


Fig. 13. Correlation traces of all partial products $r_i * d_0$ using 2 000 EM traces

traces. However, there exist also five other key hypotheses which result in a high correlation[2]. These are 3802, 7604, 15208, 30416, and 60832 (marked as black lines in the figures). Obviously, these values have the same bit representation as 1901 but are gradually shifted to the left. This is due to the fact that integer multiplication is a linear function where shifted bit combinations of the correct key have a linear impact to the multiplication result.

Next, we perform the same attacks on all other partial products that involve the first private-key word d_0. Figure 12 and Figure 13 show the result of the attacks. The correlation results of the correct key-hypothesis d_0 of all partial products are plotted on top of each other. Eleven peaks are observable that occur at locations in time when the output of the partial products is stored into the internal registers of the controller. Due to the structure of the Comba multiplication-unit, the distance between these results becomes larger the more partial products are calculated. The first partial product is calculated after about 10 μs and the last one after about 140 μs. The power-analysis attacks lead to a mean correlation of 0.72. The EM attacks yielded a mean correlation of 0.22.

After revealing the promising key candidates of the first private-key word d_0, we performed attacks on all partial products that involve the second private-key word d_1. The attacks led us to two promising key candidates: 24027 and 48054. Now we perform an attack on the second result of the final multiplication product p_1. This product word involves the calculation of the first and the second private-key word. Thus, we have to test 12 promising key hypotheses. A high correlation will occur when both hypotheses are correct. Incorrect hypotheses will show no peak. In Figure 14, the result of the power-analysis attack is given using 2 000 power traces. The correct key hypotheses (drawn in black) 1901 for d_0 and 48054 for d_1 yield a high correlation while all other key hypotheses (drawn in gray) show no peak in time when the final product is stored into the internal register of the controller. Figure 15 shows the result of the EM attack using 10 000 traces. It provides a much smaller correlation as compared to the result of the power-analysis attack. Nevertheless, the correct key can be easily discovered from the incorrect ones.

[2] The peaks do not have the same correlation value since our power model weighted the lower and higher bits of the 32-bit multiplication output differently.

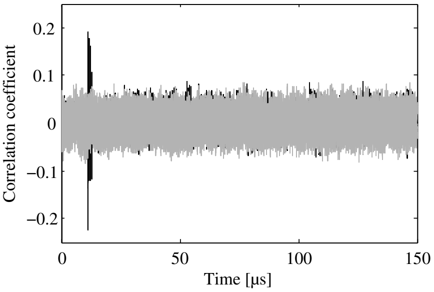


Fig. 14. Result of the power-analysis attack on the final multiplication product p_1 using 2 000 traces

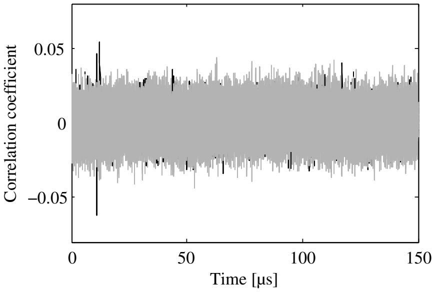


Fig. 15. Result of the EM attack on the final multiplication product p_1 using 10 000 traces

All other private-key words have been extracted by following the same strategy. Both power and electromagnetic attacks have been successful. The attacks revealed the entire private key of the ECDSA implementation which enables us to forge digital signatures and therefore to impersonate any entity and person by cloning the extracted key.

7 Conclusions

In this article, we presented the first results of DPA attacks on a hardware ECDSA implementation in a passively powered RFID device. We have designed and implemented a low-power RFID-tag prototype which consists of a 180 nm CMOS implementation of ECDSA which allows authentication by generating digital signatures. In order to evaluate the effectiveness of side-channel attacks on such devices, we performed power analysis and electromagnetic analysis. Furthermore, we propose the application of a decimation filter to reduce the complexity of the analysis on RFID devices. For both power analysis and electromagnetic analysis, the private-key could be revealed successfully. Opposed to other proposed attacks that try to reveal the ephemeral key of ECDSA, we extract the private key during signature generation. Hence, our attack is unaffected by common countermeasures that avoid the extraction of the ephemeral key. In addition, it is independent of the underlying elliptic curve representation. The significant points of our findings are as follows: First, public-key enabled RFID devices are as vulnerable to side-channel attacks as conventional contact-based devices. Second, it is not sufficient to protect the ephemeral key during scalar multiplication. It is also imperative to secure the private-key multiplication during the signature generation. This article is the first to provide results of power-analysis attacks on passive RFID devices that generate digital signatures using ECDSA. It further presents the first results of attacks on a hardware ECDSA implementation revealing the private-key during signature generation.

Future work will be to evaluate the effectiveness of this attack on existing RFID devices such as the electronic passport. The International Civil Aviation

Organization (ICAO) has published the technical specifications of e-passports in Europe and defined ECDSA as a standardized algorithm for *active authentication*. This feature allows the verification of whether the passport is authentic or not. We will evaluate the side-channel leakage of such a device to answer the question of how e-passports are effected by these attacks.

Acknowledgements

The authors would like to thank Thomas Popp and Stefan Tillich for their valuable inputs and helpful discussions. The research described in this paper has been supported by the European Commission funded project *Collaboration at Rural* under grant number 034921 (Project *C@R*) and the Austrian government funded project *CRYPTA* established under the *Trust in IT-Systems* program FIT-IT.

References

1. 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)
2. American National Standards Institute (ANSI). American National Standard X9.62-2005. Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm, ECDSA (2005)
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. Coron, J.-S.: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
5. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong Authentication for RFID Systems using the AES Algorithm. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 357–370. Springer, Heidelberg (2004)
6. 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)
7. Gebotys, C.H., Ho, S., Tiu, C.C.: EM Analysis of Rijndael and ECC on a Wireless Java-Based PDA. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 250–264. Springer, Heidelberg (2005)
8. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A Ring-Based Public Key Cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998)
9. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B., 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. Hutter, M., Mangard, S., Feldhofer, M.: Power and EM Attacks on Passive 13.56 MHz RFID Devices. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 320–333. Springer, Heidelberg (2007)

11. IEEE. IEEE Standard 1363a-2004: IEEE Standard Specifications for Public-Key Cryptography, Amendment 1: Additional Techniques (September 2004), http://ieeexplore.ieee.org/servlet/opac?punumber=9276
12. International Organisation for Standardization (ISO). ISO/IEC 7816: Identification cards - Integrated circuit(s) cards with contacts (1989)
13. International Organisation for Standardization (ISO). ISO/IEC 15693-3: Identification cards - Contactless integrated circuit(s) cards - Vicinity cards – Part 3: Anticollision and transmission protocol (2001)
14. International Organisation for Standardization (ISO). ISO/IEC 14888-3: Information technology – Security techniques – Digital signatures with appendix – Part 3: Discrete logarithm based mechanisms (2006)
15. Joye, M.: Defences Against Side-Channel Analysis. In: Advances In Elliptic Curve Cryptography. London Mathematical Society Lecture Note Series, vol. 317, pp. 87–100. Cambridge University Press, Cambridge (2005)
16. 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)
17. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
18. 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)
19. Lenstra, A.K., Verheul, E.R.: The XTR Public Key System. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 1–19. Springer, Heidelberg (2000)
20. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks – Revealing the Secrets of Smart Cards. Springer, Heidelberg (2007)
21. Medwed, M., Oswald, E.: Template Attacks on ECDSA. In: Chung, K.-I., Yung, M., Sohn, K. (eds.) 9th International Workshop on Information Security Applications (WISA 2008), Korea, Jeju Island, September 23-25, 2008, Pre-Proceedings (2008)
22. Montgomery, P.L.: Speeding the Pollard and Elliptic Curve Methods of Factorization. Mathematics of Computation 48(177), 243–264 (1987)
23. National Institute of Standards and Technology (NIST). FIPS-46-3: Data Encryption Standard (October 1999), http://www.itl.nist.gov/fipspubs/
24. National Institute of Standards and Technology (NIST). FIPS-186-2: Digital Signature Standard (DSS) (January 2000), http://www.itl.nist.gov/fipspubs/
25. National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard (November 2001), http://www.itl.nist.gov/fipspubs/
26. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard (August. 2002), http://www.itl.nist.gov/fipspubs/
27. Oren, Y., Shamir, A.: Remote Power Analysis of RFID Tags. Master's thesis, Weizmann Institute of Science, Rehovot, Israel (August 2006), http://www.wisdom.weizmann.ac.il/~yossio/rfid/
28. Plos, T.: Susceptibility of UHF RFID Tags to Electromagnetic Analysis. In: Malkin, T.G. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 288–300. Springer, Heidelberg (2008)
29. 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)
30. Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21(2), 120–126 (1978)

Author Index