Lecture Notes in Computer Science 4377

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

Editorial Board

Masayuki Abe (Ed.)

Topics in Cryptology – CT-RSA 2007

The Cryptographers' Track at the RSA Conference 2007
San Francisco, CA, USA, February 5-9, 2007
Proceedings

Volume Editor

Masayuki Abe
1-1 Hikarino-oka
Yokosuka-shi
239-0847, Japan
E-mail: abe.masayuki@lab.ntt.co.jp

Library of Congress Control Number: 2006938905

CR Subject Classification (1998): E.3, G.2.1, D.4.6, K.6.5, K.4.4, F.2.1-2, C.2, J.1

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-69327-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-69327-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11967668 06/3142 5 4 3 2 1 0

Preface

The RSA Conference, with over 15,000 attendees and 300 exhibitors, is the largest computer security event of the year. The Cryptographers' Track (CT-RSA) is a research conference within the RSA Conference. Starting in 2001, CT-RSA continues to its seventh year and is now regarded as one of the major regularly staged event for presenting the results of cryptographic research to a wide variety of audiences.

The proceedings of CT-RSA 2007 contain 25 papers selected from 73 submissions which cover all the topics of cryptography. All the submissions were reviewed by at least three reviewers, which was possible by the hard work of 23 Program Committee members and many external reviewers listed in the following pages. The papers were selected as a result of conscientious discussion. The program includes two invited talks, by Michel Rabin and Andrew Odlyzko.

I would like to express my gratitude to the Program Committee members, who were enthusiastic from the very beginning of this completed project. Thanks also to the external reviewers including those who completed urgent reviews during the discussion phase. Special thanks to Shai Halevi for providing and maintaining the Web review system. Finally, I would like to thank Burt Kaliski of RSA Laboratories and the Steering Committee for their suggestions and continuous assistance.

October 2006

Masayuki Abe
CT-RSA 2007 Program Chair

CT-RSA 2007

RSA Conference 2007, Cryptographers' Track

Moscone Center, San Francisco, CA, USA
February 5-9, 2007

Program Chair

Masayuki Abe	NTT Corporation, Japan

Program Committee

Kazumaro Aoki	NTT Corporation, Japan
John Black	University of Colorado at Boulder, USA
Colin Boyd	Queensland University of Technology, Australia
Jung Hee Cheon	Seoul National University, Korea
Alexander W. Dent	Royal Holloway, University of London, UK
Serge Fehr	CWI, Netherlands
Stuart Haber	HP Labs, USA
Shai Halevi	IBM T.J. Watson Research Center, USA
Goichiro Hanaoka	AIST, Japan
Marc Joye	Thomson R&D, France
Jonathan Katz	University of Maryland, USA
Arjen K. Lenstra	EPFL, Switzerland
Helger Lipmaa	Cybernetica AS and University of Tartu, Estonia
Stefan Lucks	University of Mannheim, Germany
Bart Preneel	Katholieke Universiteit Leuven, Belgium
Vincent Rijmen	IAIK, Graz University of Technology, Austria
Kazue Sako	NEC, Japan
Adam Smith	Weizmann Institute of Science, Israel
Douglas Stinson	University of Waterloo, Canada
Brent Waters	SRI International, USA
Susanne Wetzel	Stevens Institute of Technology, USA
Yiqun Lisa Yin	Independent Consultant, USA
Adam Young	MITRE Corporation, USA

Steering Committee

Alfred Menezes	University of Waterloo, Canada
Tatsuaki Okamoto	NTT Corporation, Japan
David Pointcheval	CNRS/ENS, France
Ron Rivest	MIT, USA
Moti Yung	RSA Labs and Columbia University, USA

External Reviewers

Toshinori Araki	Eric Hall	Ilya Mironov
Frederik Armknecht	Daewan Han	James Muir
Nuttapong Attrapadung	Helena Handschuh	Toru Nakanishi
Roberto Avanzi	Nick Hopper	DaeHun Nyang
Dan Bailey	Sotiris Ioannidis	Satoshi Obana
Lejla Batina	Toshiyuki Isshiki	Elisabeth Oswald
Liqun Chen	Tetsu Iwata	Pascal Paillier
Benoit Chevallier-Mames	Charanjit Jutla	Kenny Paterson
Andrew Clark	Ulrich Kühn	S. Raj Rajagopalan
Yvonne Cliff	Yuichi Komano	Christian Rechberger
Scott Contini	Matthias Krause	Tomas Sander
Jason Crampton	Hugo Krawczyk	Jacob Schuldt
Yang Cui	Nam-suk Kwarc	SeongHan Shin
Alex Dent	Mario Lamberger	Dirk Stegemann
Edith Elkind	Joseph Lano	Emin Tatli
Jun Furukawa	Kazuto Matsuo	Isamu Teranishi
Steven Galbraith	Alexander May	Kan Yasuda
Benedikt Gierlichs	Florian Mendel	Rui Zhang
Philippe Golle	Ulrike Meyer	

Table of Contents

Symmetric-Key Encryption

Signatures and Authentication

Hash Functions

Digital Signatures (I)

Cryptographic Protocols (I)

Side-Channel Attacks (I)

Side-Channel Attacks (II)

Cryptographic Protocols (II)

Digital Signatures (II)

Efficient Implementation

MV3: A New Word Based Stream Cipher Using Rapid Mixing and Revolving Buffers

Nathan Keller[1,⋆], Stephen D. Miller[1,2,⋆⋆], Ilya Mironov[3], and Ramarathnam Venkatesan[4]

[1] Institute of Mathematics, The Hebrew University, Jerusalem 91904, Israel
[2] Department of Mathematics, Rutgers University, Piscataway, NJ 08854
[3] Microsoft Research, 1065 La Avenida, Mountain View, CA 94043
[4] Microsoft Research, 1 Microsoft Way, Redmond, WA 98052
Cryptography, Security and Algorithms Group, Microsoft Research India

Abstract. MV3 is a new *word based* stream cipher for encrypting long streams of data. A direct adaptation of a byte based cipher such as RC4 into a 32- or 64-bit word version will obviously need vast amounts of memory. This scaling issue necessitates a look for new components and principles, as well as mathematical analysis to justify their use. Our approach, like RC4's, is based on rapidly mixing random walks on directed graphs (that is, walks which reach a random state quickly, from any starting point). We begin with some well understood walks, and then introduce nonlinearity in their steps in order to improve security and show long term statistical correlations are negligible. To minimize the short term correlations, as well as to deter attacks using equations involving successive outputs, we provide a method for sequencing the outputs derived from the walk using three revolving buffers. The cipher is fast — it runs at a speed of less than 5 cycles per byte on a Pentium IV processor. A word based cipher needs to output more bits per step, which exposes more correlations for attacks. Moreover we seek simplicity of construction and transparent analysis. To meet these requirements, we use a larger state and claim security corresponding to only a fraction of it. Our design is for an adequately secure word-based cipher; our very preliminary estimate puts the security close to exhaustive search for keys of size ≤ 256 bits.

Keywords: stream cipher, random walks, expander graph, cryptanalysis.

1 Introduction

Stream ciphers are widely used and essential in practical cryptography. Most are custom designed, e.g. alleged RC4 [Sch95, Ch. 16], SEAL [RC98], SCREAM [HCJ02], and LFSR-based NESSIE submissions such as LILI-128, SNOW, and SOBER [P+03, Ch. 3]. The VRA cipher [ARV95] has many provable properties, but requires more

⋆ Partially supported by the Adams Fellowship.
⋆⋆ Partially supported by NSF grant DMS-0301172 and an Alfred P. Sloan Foundation Fellowship. Corresponding author (miller@math.rutgers.edu).

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 1–19, 2007.

memory than the rest. We propose some new components and principles for stream cipher design, as well as their mathematical analysis, and present a concrete stream cipher called MV3.

To motivate our construction, we begin by considering RC4 in detail. It is an exceptionally short, byte-based algorithm that uses only 256 bytes of memory. It is based on random walks (card shuffles), and has no serious attacks. Modern personal computers are evolving from 32 to 64 bit words, while a growing number of smaller devices have different constraints on their word and memory sizes. Thus one may desire ciphers better suited to their architectures, and seek designs that scale nicely across these sizes. Here we focus on scaling up such random walk based ciphers. Clearly, a direct adaptation of RC4 would require vast amounts of memory.

The security properties of most stream ciphers are not based on some hard problem (e.g., as RSA is based on factoring). One would expect this to be the case in the foreseeable future. Nevertheless, they use components that – to varying degrees – are analyzable in some idealized sense. This analysis typically involves simple statistical parameters such as cycle length and mixing time. For example, one idealizes each iteration of the main loop of RC4 as a step in a random walk over its state space. This can be modeled by a graph G with nodes consisting of S_{256}, the permutations on 256 objects, and edges connecting nodes that differ by a transposition. Thus far no serious deviations from the random walk assumptions are known. Since storing an element of $S_{2^{32}}$ or $S_{2^{64}}$ is out of the question, one may try simulations using smaller permutations; however, this is nontrivial if we desire both competitive speeds and a clear analysis. It therefore is attractive to consider other options for the underlying graph G.

One of the most important parameters of RC4 is its mixing time. This denotes the number of steps one needs to start from an arbitrary state and achieve uniform distribution over the state space through a sequence of independent random moves. This parameter is typically not easy to determine. Moreover, RC4 keeps a loop counter that is incremented modulo 256, which introduces a memory over 256 steps. Thus its steps are not even Markovian (where a move from the current state is independent of earlier ones). Nevertheless, the independence of moves has been a helpful idealization (perhaps similar to Shannon's random permutation model for block ciphers), which we will also adhere to.

We identify and focus on the following problems:

- **Problem 1 – Graph Design.** How to design graphs whose random walks are suitable for stream ciphers that work on arbitrary word sizes.
- **Problem 2 – Extraction.** How to extract bits to output from (the labels of) the nodes visited by walk.
- **Problem 3 – Sequencing.** How to sequence the nodes visited by the walk so as to diminish any attacks that use relationships (e.g. equations) between successive outputs.

We now expand on these issues. At the outset, it is important to point out the desirability of simple register operations, such as additions, multiplications,

shifts, and XOR's. These are crucial for fast implementation, and preclude us from using many existing constructions of expander graphs (such as those in [LPS86, HLW06]). Thus part of the cipher design involves new mathematical proofs and constructions. The presentation of the cipher does not require these details, which may be found in the Appendix.

High level Design Principles: Clearly, a word based cipher has to output more bits per step of the algorithm. But this exposes more relationships on the output sequence, and to mitigate its effect we increase the state size and aim at security that is only a fraction of the log of the state size. We also tried to keep our analysis as transparent and construction as simple as possible. Our key initialization is a bit bulky and in some applications may require further simplifications, a topic for future research.

1.1 Graph Design: Statistical Properties and Non-linearities

In the graph design, one wants to keep the mixing time τ small as a way to keep the long term correlations negligible. This is because many important properties are guaranteed for walks that are longer than τ. For example, such a walk visits any given set S nearly the expected number of times, with exponentially small deviations (see Theorem A.2). A corollary of this fact is that each output bit is unbiased.

Thus one desires the optimal mixing time, which is on the order of $\log N$, N being the size of the underlying state space. Graphs with this property have been well studied, but the requirements for stream ciphers are more complicated, and we are not aware of any work that focuses on this issue. For example, the graphs whose nodes are $\mathbb{Z}/2^n\mathbb{Z}$ (respectively $(\mathbb{Z}/2^n\mathbb{Z})^*$) and edges are $(x, x+g_i)$ (respectively $(x, x \cdot g_i)$), where g_i are randomly chosen and $i = O(n)$, have this property [AR94]. While these graphs are clearly very efficient to implement, their commutative operations are quite linear and hence the attacks mentioned in Problem 3 above can be effective.

To this end, we introduce some nonlinearities into our graphs. For example, in the graph on $\mathbb{Z}/2^n\mathbb{Z}$ from the previous paragraph, we can also add edges of the form (x, hx) or (x, x^r). This intuitively allows for more randomness, as well as disrupting relations between successive outputs. However, one still needs to prove that the mixing time of such a modified graph is still small. Typically this type of analysis is hard to come by, and in fact was previously believed to be false. However, we are able to give rigorous proofs in some cases, and empirically found the numerical evidence to be stronger yet in the other cases. More details can be found in the Appendix.

Mixing up the random walks on multiplicative and additive abelian groups offers a principled way to combine with nonlinearities for an effective defense. As a practical matter, it is necessary to ensure that our (asymptotic) analysis applies when parameters are small, which we have verified experimentally.

We remark here that introduction of nonlinearities was the main motivation behind the construction of the T-functions of Klimov and Shamir ([KS02]). They

showed that the walk generated by a T-function deterministically visits every n-bit number once before repeating. A random walk does not go through all the nodes in the graph, but the probability that it returns to a previous node in m steps tends to the uniform probability at a rate that drops exponentially in m. It also allows us to analyze the statistical properties as indicated above. (See the Appendix for more background.)

1.2 Extraction

Obviously, if the nodes are visited truly randomly, one can simply output the LSB's of the node, and extraction is trivial. But when there are correlations among them, one can base an attack on studying equations involving successive outputs. One solution to this problem is to simultaneously hash a number of successive nodes using a suitable hashing function, but this will be expensive since the hash function has to work on very long inputs.

Our solution to the sequencing problem below allows us to instead hash a linear combination of the nodes in a faster way. A new aspect of our construction is that our hash function itself evolves on a random walk principle. We apply suitable rotations on the node labels (to alter the internal states) at the extraction step to ensure the top and bottom half of the words mix well.

1.3 Sequencing

As we just mentioned, the sequencing problem becomes significant if we wish to hash more bits to the output (in comparison to RC4). First we ensure that our graph is directed and has no short cycles. But this by itself is insufficient, since nodes visited at steps in an interval $[t, t + \Delta]$, where $\Delta \ll \tau$, can have strong correlations. Also, we wish to maximize the number of terms required in equations involved in the attacks mentioned in Problem 3. To this end, we store a short sequence of nodes visited by the walk in buffers, and sequence them properly. The buffers ensure that any relation among output bits is translated to a relation involving many nonconsecutive bits of the internal state. Hence, such relations cannot be used to mount efficient attacks on the internal state of the cipher.

The study of such designs appear to be of independent interest. We are able to justify their reduction of correlations via a theorem of [CHJ02] (see Section 4.5).

1.4 Analysis and Performance

We do not have a full analysis of the exact cipher that is implemented. However, we have ensured that our idealizations are in line with the ones that allow RC4 be viewed via random walks. Of course some degree of idealization is necessary because random bits are required to implement any random walk; here our design resembles that of alleged RC4 [Sch95, Ch. 16]. Likewise, our cipher involves combining steps from different, independent random walks on the same

underlying graph. We are able to separately analyze these processes, but although combining such steps should intuitively only enhance randomness, our exact mathematical models hold only for these separate components and hence we performed numerical tests as well.

Our cipher MV3 is fast on 32 bit processors — it runs at a speed of 4.8 cycles a byte on Pentium IV, while the speed of RC4 is about 10 cycles a byte. Only two of the eSTREAM candidates [DC06] are faster on similar architecture.

We evaluated it against some known attacks and we present the details in Section 4. We note that some of the guess-and-determine attacks against RC4 (e.g. [K+98]) are also applicable against MV3. However, the large size of the internal state of MV3 makes these attacks much slower than exhaustive key search, even for very long keys.

The security claim of MV3 is that no attack faster than exhaustive key search can be mounted for keys of length up to 256 bits.[1]

The paper is organized as follows: In Section 2 we give a description of MV3. Section 3 contains the design rationale of the cipher. In Section 4 we examine the security of MV3 with respect to various methods of cryptanalysis. Finally, Section 5 summarizes the paper. We have also included an appendix giving some mathematical and historical background. Additional appendices can be found in the full version of this paper (http://arxiv.org/abs/cs/0610048).

2 The Cipher MV3

In this section we describe the cipher algorithm and its basic ingredients. The letters in its name stand for "multi-vector", and the number refers to the three revolving buffers that the cipher is based upon.

Internal state. The main components of the internal state of MV3 are three revolving buffers A, B, and C of length 32 double words (unsigned 32-bit integers) each and a table T that consists of 256 double words. Additionally, there are publicly known indices i and u ($i \in [0 \dots 31]$, $u \in [0 \dots 255]$), and secret indices j, c, and x (c, x are double words, j is an unsigned byte).

Every 32 steps the buffers shift to the left: $A \leftarrow B$, $B \leftarrow C$, and C is emptied. In code, only the pointers get reassigned (hence the name "revolving", since the buffers are circularly rotated).

Updates. The internal state of the cipher gets constantly updated by means of pseudo-random walks. Table T gets refreshed one entry every 32 steps, via application of the following two operations:

$$u \leftarrow u + 1$$
$$T[u] \leftarrow T[u] + (T[j] \ggg 13).$$

(Symbol $x \ggg a$ means a circular rotation to the right of the double word x by a bits).

[1] Note that MV3 supports various key sizes of up to 8192 bits. However, the security claims are only for keys of size up to 256 bits.

In other words, the u-th element of the table, where u sweeps through the table in a round-robin fashion, gets updated using $T[j]$.

In its turn, index j walks (in every step, which can be idealized as a random walk) as follows:

$$j \leftarrow j \ + (B[i] \bmod 256),$$

where i is the index of the loop. Index j is also used to update x:

$$x \leftarrow x + T[j],$$

which is used to fill buffer C by $C[i] \leftarrow (x \ggg 8)$.

Also, every 32 steps the multiplier c is additively and multiplicatively refreshed as follows:

$$\begin{aligned} &c \leftarrow c + (A[0] \ggg 16)\\ &c \leftarrow c \vee 1\\ &c \leftarrow c^2 \ \text{ (can be replaced by } \ c \leftarrow c^3) \end{aligned}$$

Main loop. The last ingredient of the cipher (except for the key setup) is the instruction for producing the output. This instruction takes the following form:

$$\text{output:} \qquad (x \cdot c) \oplus A[9i+5] \oplus (B[7i+18] \ggg 16).$$

The product $x \cdot c$ of two 32-bit numbers is taken modulo 2^{32}.

Putting it all together, the main loop of the cipher is the following:

Input: length *len*
Output: stream of length *len*
repeat *len*/32 **times**
for $i = 0$ **to** 31
$j \leftarrow j + (B[i] \bmod 256)$
$x \leftarrow x + T[j]$
$C[i] \leftarrow (x \ggg 8)$
output $(x \cdot c) \oplus A[9i+5] \oplus (B[7i+18] \ggg 16)$
end for
$u \leftarrow u + 1$
$T[u] \leftarrow T[u] + (T[j] \ggg 13)$
$c \leftarrow c + (A[0] \ggg 16)$
$c \leftarrow c \vee 1$
$c \leftarrow c^2$ (can be replaced by $c \leftarrow c^3$)
$A \leftarrow B$, $B \leftarrow C$
end repeat

Key initialization. The key initialization algorithm accepts as inputs a key K of length *keylength*, which can be any multiple of 32 less than or equal to 8192 (we recommend at least 96 bits), and an initial vector IV of the same length as the key. The key remains the same throughout the entire encryption

session, though the initial vector changes occasionally. The initial vector is publicly known, but should not be easily predictable. For example, it is possible to start with a "random" IV using a (possibly insecure) pseudo-random number generator known to the attacker, and then increment the IV by 1 every time (see Section 4.2).

The key initialization algorithm is the following:

Input: key *key* and initial vector IV, both of length *keylength* double words
Output: internal state that depends on the key and the IV

$j, x, u \leftarrow 0$
$c \leftarrow 1$
fill A, B, C, T with $0x\texttt{EF}$
for $i = 0$ **to** 3
  for $l = 0$ **to** 255
    $T[i + l] \leftarrow T[i + l] + (key[l \bmod keylength] \ggg 8i) + l.$
  end for
  produce 1024 bytes of MV3 output
  encrypt T with the resulting key stream
end for
for $i = 4$ **to** 7
  for $l = 0$ **to** 255
    $T[i + l] \leftarrow T[i + l] + (IV[l \bmod keylength] \ggg 8i) + l.$
  end for
  produce 1024 bytes of MV3 output
  encrypt T with the resulting key stream
end for

Note that when only the IV is changed, only the second half of the key initialization is performed.

3 Design Rationale

In this section we describe more of the motivating principles behind the new cipher.

Internal state. The internal state of the cipher has a huge size of more than 11,000 bits. This makes guess-and-determine attacks on it (like the attack against RC4 in [K+98]) much slower than exhaustive key search, even for very long keys. In addition, it also secures the cipher from time/memory tradeoff attacks trying to invert the function $f : State \longrightarrow Output$, even for large key sizes. More detail on the security of the cipher with respect to these attacks appears in Section 4.

The buffers A, B, C and table T, as well as the indices j, c, and x should never be exposed. Since the key stream is available to the attacker and depends on this secret information, the cipher strictly adheres to the following design principles:

Principle 1. Output words must depend on as many secret words as possible.
Principle 2. Retire information faster than the adversary can exploit it.

As the main vehicle towards these goals, we use random walks (or, more precisely, pseudo-random walks, as the cipher is fully deterministic).

Updates. The updates of the internal state are based on several simultaneously applied random walks. On the one hand, these updates are very simple and can be efficiently implemented. On the other hand, as shown in the Appendix, the update mechanism allows one to mathematically prove some randomness properties of the sequence of internal states. Note that the random walks are interleaved, and the randomness of each one of them relies on the randomness of the others. Note also that the updates use addition in $\mathbb{Z}/2^n\mathbb{Z}$ and not a bitwise XOR operation. This partially resolves the problem of high-probability short correlations in random walks: In an undirected random walk, there is a high probability that after a short number of steps the state returns to a previous state, while in a directed random walk this phenomenon does not exist. For example, if we would use an update rule $x \leftarrow x \oplus T[j]$, then with probability 2^{-8} (rather than the trivial 2^{-32}) x would return to the same value after two steps. The usage of addition, which unlike XOR is not an involution, prevents this property. However, in the security proof for the idealized model we use the undirected case, since the known proofs of rapid mixing (like the theorem of Alon and Roichman [AR94]) refer to that case.

Introducing nonlinearity. In order to introduce some nonlinearity we use a multiplier c that affects the cipher output in a multiplicative way. The value of c is updated using an expander graph which involves both addition and multiplication, as explained in the Appendix. It is far from clear the squaring or cubing operation still leaves the mixing time small and our theorem addresses this.

Our update of c involves a step $c \leftarrow c \vee 1$. This operation may at a first seem odd, since it leaks $\text{LSB}(c)$ to attacker, who may use it for a distinguishing attack based only on the LSB of outputs, ignoring c entirely. However, this operation is essential, since otherwise the attacker can exploit cases where $c = 0$, which occur with a relatively high probability of 2^{-16} due to the $c \leftarrow c^2$ operation (and last for 32 steps at a time). In this situation, they can disregard the term $x \cdot c$ and devise a guess-and-determine attack with a much lower time complexity than the currently possible one.

Sequencing rule. The goals of this step were explained in section 1.3. Our output rule is based on the following general structure: The underlying walk $x_0, x_1, \ldots, x_n, \ldots$ is transformed into the output $y_0, y_1, \ldots, y_n, \ldots$ via a linear transformation:

$$y_i \quad = \quad x_{n_{i1}} \oplus x_{n_{i2}} \oplus \cdots \oplus x_{n_{ik}}.$$

Without loss of generality, we assume that the indices are sorted $n_{i1} < n_{i2} < \cdots < n_{ik}$. Let $\mathcal{N} = \{n_{ij}\}$. The set $\mathcal{N}$ is chosen to optimize the following parameters:

1. Minimize the latency and the buffer size required to compute y_i. To this end, we require that there will be two constants m and C, between 64 and 256, such that $i - C \leq n_{ij} \leq i$ for each $i \geq m$ and $1 \leq j \leq k$. We additionally constrain $n_{ik} = i$ for all $i > m$;
2. Maximize the minimal size of a set of pairs x_i, x_{i+1} that can be expressed as a linear combination of y's. More precisely, we seek to maximize a such that the following holds for some $j_1, \ldots, j_b > m$ and $i_1, \ldots, i_a$:

$$(x_{i_1} \oplus x_{i_1+1}) \oplus (x_{i_2} \oplus x_{i_2+1}) \oplus \cdots \oplus (x_{i_a} \oplus x_{i_a+1}) \;=\; y_{j_1} \oplus y_{j_2} \oplus \cdots \oplus y_{j_b}. \quad (3.1)$$

   Notice that the value of b has not been constrained, since usually this value is not too high and the attacker can obtain the required data.

Intuitively speaking, the second constraint ensures that if the smallest feasible a is large enough, no linear properties of the x walk propagate to the y walk. Indeed, any linear function on the y walk can be expressed as a function on the x walk. Since the x walk is memoryless, any linear function on a subset of x's can be written as a XOR of linear functions on the intervals of the walk. Each such interval can in turn be broken down as a sum of pairs. If a is large enough, no linear function can be a good distinguisher. Note that we concentrate on the relation between consecutive values of the state x, since in a directed random walk such pairs of states seem to be the most correlated ones.

Constructing the set $\mathcal{N}$ can be greatly simplified if $\mathcal{N}$ has periodic structure. Experiments demonstrate that for sequences with period 32 and $k = 3$, a can be as large as 12. Moreover, the best sequences have a highly regular structure, such as $n_{i1} = i - (5k \bmod 16)$ and $n_{i2} = i - 16 - (3k \bmod 16)$, where $k = i \bmod 16$. For larger periods a cannot be computed directly; an analytical approach is desirable.

As soon as the set of indices is fixed, y_i for $i > m$ can be output once x_i becomes available. The size of the buffer should be at least $i - n_{ij}$ for any $i > m$ and j. If $\mathcal{N}$ is periodic, retiring older elements can be trivially implemented by keeping several buffers and rotating between them. We note that somewhat similar buffers where used recently in the design of the stream cipher PY [BS05].

More precisely, if we choose the period $P = 32$ and $k = 3$, i.e. every output element is an XOR of three elements of the walk, the output rule can be implemented by keeping three P-word buffers, A, B, and C. Their content is shifted to the left every P cycles: A is discarded, B moves to A, and C moves to B. The last operation can be efficiently implemented by rotating pointers to the three buffers.

The exact constants chosen for n_{ij} in the output rule are chosen to maximize the girth and other useful properties of the graph of dependencies between internal variables and the output, which is available to the attacker.

Rotations. Another operation used both in the output rule and in the update of the internal state is bit rotation. The motivation behind this is as follows: all the operations used in MV3 except for the rotation (that is, bitwise XOR, modular addition and multiplication) have the property that in order to know the k least

significant bits of the output of the operation, it is sufficient to know the k least significant bits of the input. An attacker can use this property to devise an attack based on examining only the k least significant bits of the output words, and disregard all the other bits. This would dramatically reduce the time complexity of guess-and-determine attacks. For example, if no rotations were used in the cipher, then a variant of the standard guess-and-determine attack presented in Section 4 would apply. This variant examines only the least significant byte of every word, and reduces the time complexity of the attack to the fourth root of the original time complexity.

One possible way to overcome this problem is to use additional operations that do not have this problematic property, like multiplication in some other modular group. However, such operations slow the cipher significantly. The rotations used in MV3 can be efficiently implemented and prevent the attacker from tracing only the several least significant bits of the words. We note that similar techniques were used in the stream cipher SOSEMANUK [B+05] and in other ciphers as well.

Key setup. Since the bulk of the internal state is the table T, we concentrate on intermingling T and the pair (key, IV). Once T is fully dependent on the key and the IV, the revolving buffers and other internal variables will necessarily follow suit.

We have specified that the IV be as long as the key in order to prevent time/memory tradeoff attacks that try to invert the function $g : (key, IV) \longrightarrow Output$. The IV is known to the attacker but should not be easily predictable. One should avoid initializing the IV to zero at the beginning of every encryption session (as is frequently done in other applications), since this reduces the effective size of the IV and allows for better time/memory tradeoff attacks. A more comprehensive study of the security of MV3 with respect to time/memory tradeoff attacks is presented in Section 4.

We note that the key initialization phase is relatively slow. However, since the cipher is intended for encrypting long streams of data, the fast speed of the output stream generation compensates for it. We note that since the IV initialization phase is also quite slow, the IV should not be re-initialized too frequently.

4 Security

MV3 is designed to be a fast and very secure cipher. We are not aware of any attacks on MV3 faster than exhaustive key search even for huge key sizes of more than 1000 bits (except for the related key attacks in Section 4.6), but have only made security claims up to a 256-bit key size. In this section we analyze the security of MV3 against various kinds of cryptanalytic attacks.

4.1 Tests

We ran the cipher through several tests. First, we used two well-known batteries of general tests. One is Marsaglia's time-tested DIEHARD collection [Mar97],

and the other is the NIST set of tests used to assess AES candidates [R+01] (with corrections as per [KUH04]). Both test suites were easily cleared by MV3.

In light of attacks on the first few output bytes of RC4 [MS01, Mir02], the most popular stream cipher, we tested the distribution of the initial double words of MV3 (by choosing a random 160-bit key and generating the first double word of the output). No anomalies were found.

RC4's key stream is also known to have correlations between the least significant bits of bytes one step away from each other [Gol97]. Neither of the two collections of tests specifically targets bits in similar positions of the output's double words. To compensate for that, we ran both DIEHARD and NIST's tests on the most and the least significant bits of 32-bit words of the key stream. Again, none of the tests raised a flag.

4.2 Time/Memory/Data Tradeoff Attacks

There are two main types of TMDTO (time/memory/data tradeoff) attacks on stream ciphers.

The first type consists of attacks that try to invert the function $f : State \longrightarrow Output$ (see, for example, [BS00]). In order to prevent attacks of this type, the size of the internal state should be at least twice larger than the key length. In MV3, the size of the internal state is more than 11,000 bits, and hence there are no TMDTO attacks of this type faster than exhaustive key search for keys of less than 5,500 bits length. Our table sizes are larger than what one may expect to be necessary to make adequate security claims, but we have chosen our designs so that we can keep our analysis of the components transparent, and computational overhead per word of output minimal. We intend to return to this in a future paper and propose an algorithm where the memory is premium, based on different principles for light weight applications.

The second type consists of attacks that try to invert the function $g : (Key, IV) \longrightarrow Output$ (see, for example, [HS05]). The IV should be at least as long as the key – as we have mandated in our key initialization – in order to prevent such attacks faster than exhaustive key search. We note again that if the IV's are used in some predictable way (for example, initialized to zero at the beginning of the encryption session and then incremented sequentially), then the effective size of the IV is much smaller, and this may enable a faster TMDTO attack. However, in order to overcome this problem the IV does not have to be "very random". The only thing needed is that the attacker will not be able to know which IV will be used in every encryption session. This can be achieved by initializing the IV in the beginning of the session using some (possibly insecure) publicly known pseudo-random number generator and then incrementing it sequentially.

4.3 Guess-and-Determine Attacks

A guess-and-determine attack against RC4 appears in [K+98]. The attack can be adapted to MV3 (the details of this modification are given in the full version of this paper). However, the time complexity of this attack is quite large – more

than 2^{2000}, since the attacker starts with guessing more than 2000 bits of the state. Hence, this attack is slower than exhaustive key search for keys of less then 2000 bits length.

4.4 Guess-and-Determine Attacks Using the Several Least Significant Bits of the Words

Most of the operations in MV3 allow the attacker to focus the attack on the k least significant bits, thus dramatically reducing the number of bits guessed in the beginning of the attack. We consider two reasonable attacks along these lines.

The first attack concentrates on the least significant bit of the output words. In this case, since the least significant bit of c is fixed to 1, the attacker can disregard c at all. However, in this case the attacker cannot trace the values of j, and guessing them all the time will require a too high time complexity. Hence, it seems that this attack is not applicable to MV3.

The second attack concentrates on the eight least significant bits of every output word. If there were no rotations in the update and output rules, the attacker would indeed be able to use her guess to trace the values of j and the eight least significant bits in all the words of the internal state. This would result in an attack with time complexity of about 2^{600}. However, the rotations cause several difficulties for such an attack, because guesses in consecutive loops cannot be combined together.

Hence, it seems that both of the attacks cannot be applied, unless the attacker guesses the full values of all the words in two buffers, which leads to the attack described in subsection 4.3 (with a time complexity of more than 2^{2000}).

4.5 Linear Distinguishing Attacks

Linear distinguishing attacks aim at distinguishing the cipher output from random streams, using linear approximations of the non-linear function used in the cipher – in our case, the random walk.

In [CHJ02], Coppersmith et al. developed a general framework to evaluate the security of several types of stream ciphers with respect to these attacks. It appears that the structure of MV3 falls into this framework, to which [CHJ02, Theorem 6] directly applies:

Theorem 1. *Let ϵ be the bias of the best linear approximation one can find for pairs x_i, x_{i+1}, and let $A_N(a)$ be the number of equations of type (3.1) that hold for the sequence $y_m, y_{m+1}, \ldots$. Then the statistical distance between the cipher and the random string is bounded from above by*

$$\sqrt{\sum_{a=1}^{N} A_N(a)\epsilon^{2a}}. \tag{4.1}$$

Note that for $\epsilon \ll 1/2$, the bound (4.1) is dominated by the term with the smallest a, which equals to 12 in our case. Since the relation between x_i and x_{i+1} is based on a random walk, ϵ is expected to be very small. Since the statistical distance is of order ϵ^{24}, we expect that the cipher cannot be distinguished from a random string using a linear attack, even if the attacker uses a very long output stream for the analysis.

4.6 Related-Key Attacks and Key Schedule Considerations

Related key attacks study the relation between the key streams derived from two unknown, but related, secret keys. These attacks can be classified into distinguishing attacks, that merely try to distinguish between the key stream and a random stream, and key recovery attacks, that try to find the actual values of the secret keys.

One of the main difficulties in designing the key schedule of a stream cipher with a very large state is the vulnerability to related-key distinguishing attacks. Indeed, if the key schedule is not very complicated and time consuming, an attacker may be able to find a relation between two keys that propagates to a very small difference in the generated states. Such small differences can be easily detected by observing the first few words of the output stream.

It appears that this difficulty applies to the current key schedule of MV3. For long keys, an attacker can mount a simple related-key distinguishing attack on the cipher. Assume that $keylength = 8192/t$. Then in any step of the key initialization phase, every word of the key affects exactly t words in the T array, after which the main loop of the cipher is run eight times and the output stream is XORed (bit-wise) to the content of the T array. The same is repeated with the IV replacing the key in the IV initialization phase.

The attacker considers encryption under the same key with two IVs that differ only in one word. Since the key is the same in the two encryptions, the entire key initialization phase is also the same. After the first step of the IV initialization, the intermediate values differ in exactly t words in the T array. Then, the main loop is run eight times. Using the random walk assumption, we estimate that, with probability $(1 - t/256)^{256}$, each of the corresponding words in the respective T arrays used in these eight loops are equal, making the output stream equal in both encryptions. Hence, with probability $(1 - t/256)^{256}$, after the first step of the IV initialization the arrays A, B, and C are equal in both encryptions and the respective T arrays differ only in t words.

The same situation occurs in the following three steps of the IV initialization. Therefore, with probability $\prod_{\ell=1}^{4}(1 - t\ell/256)^{256}$ all of the corresponding words used during the entire initialization phase are equal in the two encryptions. Then with probability $(1 - 4t/256)^{32}$ all of the corresponding words used in the first loop of the key stream generation are also equal in the two encryptions, resulting in two equal key streams. Surely this can be easily recognized by the attacker after observing the key stream generated in the first loop.

In order to distinguish between MV3 and a random cipher, the attacker has to observe about $M = \prod_{\ell=1}^{4}(1 - t\ell/256)^{-256} \cdot (1 - 4t/256)^{-32}$ pairs of related IVs, and for each pair she has to check whether there is equality in the first 32 key stream words. Hence, the data and time complexities of the attack are about $2^{10}M$. For keys of length at least 384 bits, this attack is faster than exhaustive key search. Note that (somewhat counter intuitively) the attack becomes more efficient as the length of the key is increased. The attack is most efficient for 8192-bit keys, where the data complexity is about 2^{10} bits of key stream encrypted under the same key and 2^{15} pairs of related IVs, and the time complexity is less than 2^{32} cycles. For keys of length at most 256 bits, the data and time complexities of the attack are at least 2^{618} and hence the related-key attack is much slower than exhaustive key search.

If we try to speed up the key schedule by reducing the number of loops performed at each step of the key schedule, or by inserting the output of the eight loops into the T array (as opposed to XORing it bit-wise to the content of the T array), the complexity of the related-key attack is reduced considerably. These details are given in the full version of the paper.

Hence, the related-key attack described above is a serious obstacle to speeding up the key schedule. However, we note that the related-key model in general, and in particular its requirement of obtaining a huge number of encryptions under different related-IV pairs, is quite unrealistic.

4.7 Other Kinds of Attacks

We subjected the cipher to other kinds of attacks, including algebraic attacks and attacks exploiting classes of weak keys. We did not find any discrepancies in these cases.

5 Summary

We have proposed a new fast and secure stream cipher, MV3. The main attributes of the cipher are efficiency in software, high security, and its basis upon clearly analyzable components.

The cipher makes use of new rapidly mixing random walks, to ensure the randomness in the long run. The randomness in the short run is achieved by revolving buffers that are easily implemented in software, and break short correlations between the words of the internal state.

The cipher is word-based, and hence is most efficient on 32-bit processors. On a Pentium IV, the cipher runs with a speed of 4.8 clocks a byte.

Acknowledgements. We thank Adi Shamir for his generous discussions. We are grateful to Nir Avni and Uzi Vishne for their careful reading and comments on an earlier version, and Rebecca Landy for providing numerics on expander graphs.

References

[ARV95] W. Aiello, S. Rajagopalan, and R. Venkatesan, *Design of Practical and Provably Good Random Number Generators*, Proc. of SODA'95, pp. 1–9, 1995.

[AR94] N. Alon and Y. Roichman, *Random Cayley Graphs and Expanders*, Rand. Str. Alg. vol. 5(2), pp. 271–284, 1994.

[B+05] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert, *Sosemanuk, a Fast Software-Oriented Stream Cipher*, submitted to Ecrypt, 2005.

[BS05] E. Biham and J. Seberry, *Py (Roo): A fast and secure stream cipher using rolling arrays*, submitted to Ecrypt, 2005.

[BS00] A. Biryukov and A. Shamir, *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*, Proc. of Asiacrypt'00, pp. 1–13, 2000.

[CHJ02] D. Coppersmith, S. Halevi, and C. S. Jutla, *Cryptanalysis of stream ciphers with linear masking*, Proc. of CRYPTO'02, pp. 515–532, 2002.

[DC06] C. De Canniere, *eSTREAM testing framework*, 2006, avalable on-line at http://www.ecrypt.eu.org/stream.

[Gi98] D. Gillman, *A Chernoff bound for random walks on expander graphs*, SIAM J. Comput. 27 (4), pp.1203–1220 (1998).

[Gol97] J. Golić, *Linear statistical weakness of alleged RC4 keystream generator*, Proc. of Eurocrypt'97, pp. 226–238, 1997.

[HCJ02] S. Halevi, D. Coppersmith, and C. Jutla, *Scream: a Software-Efficient Stream Cipher*, Proc. of FSE'02, pp. 195–209, 2002.

[HS05] J. Hong, P. Sarkar, *Rediscovery of Time Memory Tradeoffs*, 2005. Available online at http://eprint.iacr.org/2005/090.

[HLW06] S. Hoory, N. Linial, and A. Wigderson, *Expander graphs and their applications*, Bull. Amer. Math. Soc. **43** (2006), pp. 439–561.

[JMV05] D. Jao, S. Miller, and R. Venkatesan, *Do All Elliptic Curves of the Same Order Have the Same Difficulty of Discrete Log?*, Proc. of Asiacrypt'05, pp. 21–40, 2005.

[KUH04] S. Kim, K. Umeno, and A. Hasegawa, *Corrections of the NIST statistical test suite for randomness*, Cryptology ePrint Archive, Report 2004/018, 2004.

[KS02] A. Klimov and A. Shamir, *A New Class of Invertible Mappings*, Proc. of CHES'02, pp. 470–483, 2002.

[KS04] A. Klimov and A. Shamir, *New Cryptographic Primitives Based on Multiword T-Functions*, Proc. of FSE'04, pp. 1–15, 2004.

[K+98] L. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege, *Analysis Methods for (Alleged) RC4*, Proc. of ASIACRYPT'98, pp.327-341, 1998.

[LPS86] A. Lubotzky, R. Phillips, and P. Sarnak, *Explicit expanders and the Ramanujan conjectures*, Proc. of STOC'86, pp. 240–246, 1986.

[MS01] I. Mantin and A. Shamir, *A practical attack on broadcast RC4*, Proc. of FSE'01, pp. 152–164, 2001.

[Mar97] G. Marsaglia, *DIEHARD battery of tests*, available from `http://stat.fsu.edu/~geo/`.

[Mir02] I. Mironov, *(Not so) random shuffles of RC4*, Proc. of CRYPTO'02, pp. 304–319, 2002.

[P+03] B. Preneel et al., *NESSIE Security Report, version 2.0*, 2003.

[RC98] P. Rogaway and D. Coppersmith, *A Software-Optimized Encryption Algorithm*, J. of Cryptology 11(4), pp. 273–287, 1998.

[R+01] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, NIST Special Publication 800-22, `http://www.nist.gov`, 2001.

[Sch95] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd Ed., John Wiley & Sons, 1995.

A Appendix: Mathematical Background

The good long term randomness properties of the internal state of MV3 are achieved by updates using rapidly mixing random walks. Actually, the walks are only pseudo-random since the cipher is fully deterministic, but we desire the update rule to be as close as possible to a random walk. In this appendix we recall some mathematics used to study random walks, such as expander graphs and the rapid mixing property, with a focus on the types of random walks used in the cipher.

Rapidly Mixing Random Walks and Expander Graphs

Recall that a random walk on a graph starts at a node z_0, and at each step moves to a node connected by one of its adjacent edges at random. A lazy random walk is the same, except that it stays at the same node with probability $1/2$, and otherwise moves to an adjacent node at random. Intuitively, a random walk is called "rapidly mixing" if, after a relatively short time, the distribution of the state of the walk is close to the uniform distribution — regardless of the initial distribution of the walk.

Next, we come to the notion of expander graph. Let Γ be an undirected k-regular graph on $N < \infty$ vertices. Its adjacency operator acts on $L^2(\Gamma)$ by summing the values of a function at the neighbors of a given vertex:

$$(Af)(x) \quad = \quad \sum_{x \sim y} f(y)\,. \tag{A.1}$$

The spectrum of A is contained in the interval $[-k, k]$. The *trivial eigenvalue* $\lambda = k$ is achieved by the constant eigenvector; if the graph is connected then this eigenvalue has multiplicity 1, and all other eigenvalues are strictly less than k. A sequence of k-regular graphs (where the number of vertices tends to infinity) is customarily called a sequence of *expanders* if all nontrivial eigenvalues λ of all the graphs in the sequence satisfy the bound $|\lambda| \leq k - c$ for an absolute constant c. We shall take a slightly more liberal tack here and consider graphs which satisfy the weaker eigenvalue bound $|\lambda| \leq k - c(\log N)^{-A}$ for some constant $A \geq 0$.

The importance of allowing the lenient eigenvalue bound $|\lambda| \leq k - c(\log N)^{-A}$ is that a random walk on such a graph mixes in polylog(N) time, even if $A > 0$. More precisely, we have the following estimate (see, for example, [JMV05, Proposition 3.1]).

Proposition A.1. *Let Γ be a regular graph of degree k on N vertices. Suppose that the eigenvalue λ of any nonconstant eigenvector satisfies the bound $|\lambda| \leq \sigma$ for some $\sigma < k$. Let S be any subset of the vertices of Γ, and x be any vertex in Γ. Then a random walk of any length at least $\frac{\log 2N/|S|^{1/2}}{\log k/\sigma}$ starting from x will land in S with probability at least $\frac{|S|}{2N} = \frac{|S|}{2|\Gamma|}$.*

Indeed, with $\sigma = k - c(\log N)^{-A}$, the random walk becomes evenly distributed in the above sense after $O((\log N)^{A+1})$ steps.

Next, we come to the issue of estimating the probability that the random walk returns to a previously visited node. This is very important for cryptographic purposes, since short cycles lead to relations which an attacker can exploit. The following result gives a very precise estimate of how unlikely it is that a random walk returns to the vertex it starts from. More generally, it shows that if one has *any* set S consisting of, say, one quarter of all nodes, then the number of visits of the random walk to this set will be exceptionally close to that of a purely random walk in the sense that it will obey a Chernoff type bound. This in turn allows one to show that the idealized cipher passes all the moment tests.

Theorem A.2. *([Gi98, Theorem 2.1]) Consider a random walk on a k-regular graph Γ on N vertices for which the second-largest eigenvalue of the adjacency operator A equals $k - \varepsilon k$, $\varepsilon > 0$. Let S be a subset of the vertices of Γ, and t_n the random variable of how many times a particular walk of n steps along the graph lands in S. Then, as sampled over all random walks, one has the following estimate for any $x > 0$:*

$$\text{Prob}\left[\left|t_n - n\frac{|S|}{|\Gamma|}\right| \geq x\right] \leq \left(1 + \frac{x\varepsilon}{10n}\right) e^{-x^2\varepsilon/(20n)}. \tag{A.2}$$

Thus even with a moderately small value of ε, the random walk avoids dwelling in any one place overly long.

In practice, algorithms often actually consider random walks on *directed* graphs. The connection between rapid mixing of directed graphs (with corresponding adjacency/transition matrix M) and undirected graphs is as follows. A result of J. Fill shows that if the additive reversalization (whose adjacency matrix is $M + M^t$) or multiplicative reversalization (whose adjacency matrix is MM^t) rapidly mixes, then the lazy random walk on the directed version also rapidly mixes. From this it is easy to derive the effect of having no self-loops as well. Moreover, if the undirected graph has expansion, then so does the directed graph — provided it has an Eulerian orientation. It is important to note that this implication can also be used to greatly improve poorly mixing graphs, which is the genesis of the graph in Theorem A.3.

Expander graphs are natural sources of (pseudo)randomness, and have numerous applications as extractors, de-randomizers, etc. (see [HLW06]). However, there are a few practical problems that have to be resolved before expanders can be used in cryptographic applications. One of these, as mentioned above, is a serious security weakness: the walks in such a graph have a constant probability

of returning to an earlier node in constant number of steps. It is possible to solve this problem by adding the current state (as a binary string) to that of another process which has good short term properties, but this increases the cache size. In addition, if the graph has large directed girth (i.e. no short cycles), then the short term return probabilities can be minimized or even eliminated.

Additive Random Walks on $\mathbb{Z}/2^n\mathbb{Z}$

Most of the random walks used in the cipher, namely the random walks used in the updates of j, x, and T, are performed in the additive group $\mathbb{Z}/2^n\mathbb{Z}$. The mixing properties of these walks can be studied using results on *Cayley graphs* of this group. In general, given a group G with a set of generators S, the Cayley graph $X(G, S)$ of G with respect to S is the graph whose vertices consist of elements of G, and whose edges connect pairs (g, gs_i), for all $g \in G$ and $s_i \in S$.

Alon and Roichman [AR94] gave a detailed study of the expansion properties of abelian Cayley graphs, viewed as undirected graphs. They showed that $X(G, S)$ is an expander when S is a randomly chosen subset of G whose size is proportional to $\log |G|$, and hence random walks on them mix rapidly on them.

Using second-moment methods it can be shown that their graphs are ergodic (and also that the length of the shortest cycle is within a constant factor of $\log |\Gamma|$) with overwhelming probability over the choice of generators. The significance of this is that we need not perform a lazy random walk, which would introduce undesirable short term correlations as well as waste cycles and compromise the cryptographic strength.

In MV3, the rapid mixing of the random walks updating x, j and T follows from the theorem of Alon and Roichman. For example, consider the update rule of x of the form $x \leftarrow x + T[j]$. The update rule corresponds to a random walk on the Cayley graph $X(G, S)$ where G is the additive group $\mathbb{Z}/2^n\mathbb{Z}$ and S consists of the 256 elements of the T register. Note that we have $|S| = 4 \log_2(|G|)$. In order to apply the theorem of Alon and Roichman we need that the elements of the T array will be random and that the walk will be random, that is, that j will be chosen each time randomly in $\{0, \ldots, 255\}$. Hence, assuming that j and T are uniformly distributed, we have a rapid mixing property for x. Similarly, one can get rapid mixing property for j using the randomness of x.

Non-linear Random Walks

In order to introduce some nonlinearity to the cipher, we use a multiplier c that affects the cipher output in a multiplicative way. The multiplier itself is updated using a nonlinear random walk that mixes addition and multiplication operations. The idealized model of this random walk is described in the following theorem:

Theorem A.3. *Let N and r be relatively prime positive integers greater than 1, and $\bar{r}$ an integer such that $r\bar{r} \equiv 1 \pmod{N}$. Let Γ be the 4-valent graph on $\mathbb{Z}/N\mathbb{Z}$ in which each vertex x is connected to the vertices $r(x + 1)$, $r(x - 1)$, $\bar{r}x + 1$, and $\bar{r}x - 1$. Then there exists a positive constant $c > 0$, depending only on r, such that all nontrivial eigenvalues λ of the adjacency matrix of Γ either*

satisfy the bound $|\lambda| \leq 4 - c(\log N)^{-2}$*, or are of the form* $\lambda = 4\cos(2\pi k/N)$ *for* k *satisfying* $rk \equiv k \pmod{N}$*. In particular, if* N *is a power of 2 and* $(r-1, N) = 2$*, then* Γ *is a bipartite graph for which all eigenvalues not equal to* ± 4 *satisfy* $|\lambda| \leq 4 - c(\log N)^{-2}$*.*

The proof of the theorem can be found in the full version of the paper. The result means that for a fixed r, Γ is an expander graph in the looser sense that its eigenvalue separation is at least $c/(\log N)^2$ for N large. This is still enough to guarantee that the random walk on the graph mixes rapidly (i.e. in polylog(N) time).

We note that although we use an additive notation, the theorem holds for any cyclic group, for example a multiplicative group in which the multiplication by r corresponds to exponentiation (this is the non-linearity we are referring to). Also the expressions $r(x \pm 1)$, $\bar{r}x \pm 1$ may be replaced by $r(x \pm g)$, $\bar{r}x \pm g$ for any integer g relatively prime to N. Additionally, the expansion remains valid if a finite number of extra relations of this form are added.

The operation used in the MV3 cipher algorithm itself is slightly different: it involves not only addition steps, but also a squaring or cubing step. Though this is not covered directly the Theorem, it is similar in spirit. We have run extensive numerical tests and found that this operation can in fact greatly enhance the eigenvalue separation, apparently giving stronger eigenvalue bounds of the form $|\lambda| \leq \sigma$ for some constant $\sigma < 4$.

A Simple Related-Key Attack on the Full SHACAL-1

Eli Biham[1,⋆], Orr Dunkelman[1,2,⋆], and Nathan Keller[3,⋆⋆]

[1] Computer Science Department, Technion.
Haifa 32000, Israel
{biham,orrd}@cs.technion.ac.il
[2] Katholieke Universiteit Leuven, ESAT/SCD-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
orr.dunkelman@esat.kuleuven.be
[3] Einstein Institute of Mathematics, Hebrew University.
Jerusalem 91904, Israel
nkeller@math.huji.ac.il

Abstract. SHACAL-1 is a 160-bit block cipher with variable key length of up to 512-bit key based on the hash function SHA-1. It was submitted to the NESSIE project and was accepted as a finalist for the 2nd phase of evaluation. Since its introduction, SHACAL-1 withstood extensive cryptanalytic efforts. The best known key recovery attack on the full cipher up to this paper has a time complexity of about 2^{420} encryptions.

In this paper we use an observation due to Saarinen to present an elegant related-key attack on SHACAL-1. The attack can be mounted using two to eight unknown related keys, where each additional key reduces the time complexity of retrieving the actual values of the keys by a factor of 2^{62}. When all eight related-keys are used, the attack requires $2^{101.3}$ related-key chosen plaintexts and has a running time of $2^{101.3}$ encryptions. This is the first successful related-key key recovery attack on a cipher with varying round constants.

1 Introduction

In 1993, NIST has issued a standard hash function called Secure Hash Algorithm (FIPS-180) [27]. Later this version was named SHA-0, as NIST published a small tweak to this standard called SHA-1 in 1995. Both SHA-0 and SHA-1 are Merkle-Damgard hash functions with compression function that accept blocks of 512 bits and chaining values of 160 bits (which is also the digest size). Later, NIST has published three more standard hash functions as part of FIPS-180: SHA-256, SHA-384 and SHA-512. Each of the new hash functions has a digest size corresponding to its number, i.e., SHA-256 has a 256-bit digest, etc. Recently, NIST has issued another hash function, SHA-224, that has a digest size of 224 bits.

⋆ This work was supported in part by the Israel MOD Research and Technology Unit.
⋆⋆ The author was supported by the Adams fellowship.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 20–30, 2007.

Both SHA-0 and SHA-1 were subjected to a great deal of analysis [2,3,12,30, 32,34] In the last two years there was a major progress in the attacks on both of the hash functions. This progress included finding a collision in SHA-0, and devising an algorithm that can find a collision in SHA-1 in less than 2^{63} SHA-1 applications [2,3,30,32,34]. The new techniques are based on finding good differentials of the compression function of SHA-1 and combining them with some novel plaintext modification techniques.

In 2000 it was suggested to use the compression function of SHA-1 as a block cipher [15]. Later this suggestion was named SHACAL-1 and submitted to the NESSIE project [16]. SHACAL-1 is a 160-bit block cipher with a variable key length (0–512 bits) and 80 rounds based on the compression function of SHA-1. The cipher was selected as a NESSIE finalist, but was not selected for the NESSIE portfolio [25].

Several papers analyze the strength of SHACAL-1 as a block cipher [6,17,20,21]. These papers apply differential, amplified boomerang, rectangle and related-key rectangle attacks to reduced-round variants of SHACAL-1. The best known attack on SHACAL-1 that does not use related-keys is a rectangle attack on 49-round SHACAL-1 [6].

In a recent paper a transformation of the collision-producing differentials of SHA-1 presented in [32] was used to devise the first known attack on the full SHACAL-1 [13]. The attack is a related-key rectangle attack that requires $2^{159.8}$ chosen plaintexts encrypted under four related keys and has a time complexity of 2^{423} encryptions.

In [23], Saarinen observed that it is possible to construct slid pairs in the compression function of SHA-1 using about 2^{97} chosen chaining values (for two different blocks of message). Saarinen used the slid pairs to mount a related-key distinguishing attack against SHACAL-1 requiring 2^{97} chosen plaintexts encrypted under two related keys and time complexity of 2^{97} encryptions.

In this paper we use the results of [23] to devise key-recovery attacks against the full SHACAL-1 with much lower data and time complexities than previously known. The attacks use between two and eight unknown related keys, where each additional key reduces the time complexity of retrieving the actual values of the keys by a factor of 2^{62}. When all eight related-keys are used, the attack requires $2^{101.3}$ related-key chosen plaintexts and has a running time of $2^{101.3}$ encryptions. A comparison of the known attacks on SHACAL-1 along with our new results is presented in Table 1.

This is the first time a related-key attack succeeds against a cipher with varying round constants. Moreover, this is the first case, where the related-key process is used with some probability without combining it with other attacks, e.g., related-key differentials [19] or related-key rectangle attack [7,20,17].

This paper is organized as follows: In Section 2 we describe the block cipher SHACAL-1. In Section 3 we describe the previously known results on SHACAL-1. We shortly describe Saarinen's main observation on SHA-1 in Section 4. In Section 5 we present our new related-key attack on SHACAL-1. We summarize the paper in Section 6.

Table 1. Summary of Our Results and Previously Known Results on SHACAL-1

Attack & Source	Number of		Rounds	Complexity	
	Keys	Rounds		Data	Time
Differential [21]	1	41	0–40	2^{141} CP	2^{491}
Amplified Boomerang [21]	1	47	0–46	$2^{158.5}$ CP	$2^{508.4}$
Rectangle [6]	1	47	0–46	$2^{151.9}$ CP	$2^{482.6}$
Rectangle [6]	1	49	29–77	$2^{151.9}$ CC	$2^{508.5}$
Related-Key Rectangle [20]	2	59	0–58	$2^{149.7}$ RK-CP	$2^{498.3}$
Related-Key Rectangle [17]	4	70	0–69	$2^{151.8}$ RK-CP	$2^{500.1}$
Related-Key Rectangle [13]	4	80	0–79	$2^{159.8}$ RK-CP	$2^{423.0}$
Related-Key Rectangle [13]	4	80	0–79	$2^{153.8}$ RK-CP	$2^{504.2}$
Slide † [23]	2	80	0–79	2^{97} RK-CP	2^{97}
Related Key (Section 5)	2	80	0–79	2^{97} RK-CP	2^{447}
Related Key (Section 5)	4	80	0–79	$2^{99.6}$ RK-CP	2^{321}
Related Key (Section 5)	8	80	0–79	$2^{101.3}$ RK-CP	$2^{101.3}$

Complexity is measured in encryption units.
† – Distinguishing attack
CP — Chosen Plaintexts, CC — Chosen Ciphertexts, RK — Related-Key

2 Description of SHACAL-1

SHACAL-1 [16] is a 160-bit block cipher supporting variable key lengths (0–512 bits). It is based on the compression function of the hash function SHA-1 [27]. The cipher has 80 rounds (also referred as steps) grouped into four types of 20 rounds each.[1]

The 160-bit plaintext is divided into five 32-bit words – A, B, C, D and E. We denote by X_i the value of word X before the ith round, i.e., the plaintext P is divided into A_0, B_0, C_0, D_0 and E_0, and the ciphertext is composed of $A_{80}, B_{80}, C_{80}, D_{80}$ and E_{80}.

In each round the words are updated according to the following rule:

$$\begin{aligned} A_{i+1} &= W_i + ROTL_5(A_i) + f_i(B_i, C_i, D_i) + E_i + K_i \\ B_{i+1} &= A_i \\ C_{i+1} &= ROTL_{30}(B_i) \\ D_{i+1} &= C_i \\ E_{i+1} &= D_i \end{aligned}$$

where + denotes addition modulo 2^{32}, $ROTL_j(X)$ represents rotation to the left by j bits, W_i is the round subkey, and K_i is the round constant.[2] There are three different functions f_i, selected according to the round number:

[1] To avoid confusion, we adopt the common notations for rounds. In [16] the notation step stands for round, where round is used for a group of 20 steps.

[2] This time we adopt the notations of [16], and alert the reader of the somewhat confusing notations.

$$\begin{array}{ll} f_i(X,Y,Z) = f_{if} = (X\&Y)|(\neg X\&Z) & 0 \le i \le 19 \\ f_i(X,Y,Z) = f_{xor} = (X \oplus Y \oplus Z) & 20 \le i \le 39, 60 \le i \le 79 \\ f_i(X,Y,Z) = f_{maj} = ((X\&Y)|(X\&Z)|(Y\&Z)) & 40 \le i \le 59 \end{array}$$

There are also four round constants:

Rounds	K_i	Rounds	K_i
0–19	$5A827999_x$	20–39	$6ED9EBA1_x$
40–59	$8F1BBCDC_x$	60–79	$CA62C1D6_x$

In [16] it is strongly advised to use keys of at least 128 bits, even though shorter keys are supported. The first step in the key schedule algorithm is to pad the supplied key into a 512-bit key. Then, the 512-bit key is expanded into eighty 32-bit subkeys (or a total of 2560 bits of subkey material). The expansion is done in a linear manner using a linear feedback shift register (over $GF(2^{32})$).

The key schedule is as follows: Let $M_0, \ldots, M_{15}$ be the 16 key words (32 bits each). Then the round subkeys $W_0, \ldots, W_{79}$ are computed by the following algorithm:

$$W_i = \begin{cases} M_i & 0 \le i \le 15 \\ ROTL_1(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) & 16 \le i \le 79 \end{cases}$$

3 Previous Results

A preliminary differential and linear analysis of the properties of the compression function of SHA-1 as a block cipher is presented in [15]. The found differentials are relatively short (10 rounds) and have probabilities varying between 2^{-13} and 2^{-26} (depending on the round functions).

In [28] these differentials are improved, and 20-round differentials with probability 2^{-41} are presented. In [21] another set of differentials of SHACAL-1 is presented, including a 30-round differential with probability 2^{-130}.

In [21] a 21-round differential for rounds 0–20 and a 15-round differential for rounds 21–35 are combined to devise an amplified boomerang distinguisher [18] for 36-round SHACAL-1. This distinguisher is used to attack 39-round SHACAL-1 using $2^{158.5}$ chosen plaintexts and about $2^{250.8}$ 39-round SHACAL-1 encryptions. The attack is based on guessing the subkeys of the three additional rounds, and then checking whether the distinguisher succeeds. This approach is further extended to attack 47-round SHACAL-1 before exhaustive key search becomes faster than this attack. Another attack presented in [21] is a differential attack on 41-round SHACAL-1. The success of these attacks was questioned and resolved in [6].

Besides resolving the problems with previous attacks, in [6] a rectangle attack on 49-round SHACAL-1 is presented. The attack requires $2^{151.9}$ chosen plaintexts, and has a running time equivalent to $2^{508.5}$ 49-round SHACAL-1 encryptions.

In [20] a related-key rectangle attack with two keys is presented against 59-round SHACAL-1. This attack has a data complexity of $2^{149.7}$ related-key chosen

plaintexts and a time complexity of $2^{498.3}$ 59-round SHACAL-1 encryptions. This attack is improved in [17] to a related-key rectangle attack with four keys on 70-round SHACAL-1. The improved attack has a data complexity of $2^{151.8}$ related-key chosen plaintexts, and a time complexity of $2^{500.1}$ 70-round SHACAL-1 encryptions.

Using the improved differentials of SHA-1 found in [32] and some improved key recovery techniques, two related-key rectangle attacks with four keys on the full SHACAL-1 are given in [13]. The first has a data complexity of $2^{159.8}$ related-key chosen plaintexts and a time complexity of $2^{423.0}$ encryptions, and the second has a data complexity of $2^{153.8}$ related-key chosen plaintexts and a time complexity of $2^{504.2}$ encryptions.

Summarizing the known attacks on SHACAL-1, the best known attacks against SHACAL-1 use the rectangle technique. The best attack in the related-key model is applicable against the full SHACAL-1, while the best chosen plaintext attack is applicable for a 49-round reduced variant of the cipher. Both of the attacks require a very large amount of chosen plaintexts and a very high time complexity.

4 Slid Pairs in the Compression Function of SHA-1

4.1 Related-Key Attacks and Slid Pairs

Related key attacks [1,22] are attacks that exploit relations during encryption under different keys. Let us consider an iterated block cipher whose key schedule is simple enough such that for any key K_1, it is possible to find K_2 such that the round subkeys KR_i^1, KR_i^2 produced by K_1 and K_2, respectively, satisfy:

$$KR_i^1 = KR_{i+1}^2$$

Assume that like in many ciphers, all the rounds of the cipher are the same. For such pair of keys, if a pair of plaintexts (P_1, P_2) satisfies $P_1 = f_{KR_1^2}(P_2)$, where $f_{sk}(P)$ denotes one round encryption of P under the subkey sk, then the corresponding ciphertexts C_1 and C_2 satisfy $C_1 = f_{KR_r^1}(C_2)$, where r is the number of rounds. Given such a pair of plaintexts for these related keys, it is possible to find the keys using a very simple attack algorithm.

In [10] Biryukov and Wagner show that the related key attacks can be applied to ciphers that can be written as $E_k = f_k^l = f_k \circ f_k \circ \ldots \circ f_k$, where f_k is a "relatively simple" key-dependent function. The attack looks for two plaintexts P_1 and P_2 satisfying the relation $P_2 = f_k(P_1)$. Such a pair is called a *slid pair*, and can be used in an attack that is similar to the attack in the case of related keys.

In the slide attack, the attacker collects $2^{n/2}$ known plaintext/ciphertext pairs (where n is the block size). For every pair of plaintexts (P_1, P_2), the attacker checks whether it is a slid pair by treating the pair as a slid pair and trying to use it to attack f_k. The time complexity of the attack is 2^n applications of the attack on f_k given a slid pair. Note that the data and time complexities are independent of the number of rounds in E_k. The slide attack was further generalized in [8,11,14] to be applicable to a wider range of block ciphers.

4.2 Saarinen's Observation

Saarinen has observed that slid pairs can be found (with some probability) in the compression function of SHA-1, i.e., in SHACAL-1 [23]. The slid pairs are constructed under two related message blocks, or in the case of SHACAL-1, two related keys.

Let $W = (W_0, W_1, \ldots, W_{15})$ be the first key, and let $W^* = (W_0^*, W_1^*, \ldots, W_{15}^*)$, such that

$$W_i^* = \begin{cases} W_{i+1} & 0 \le i \le 14 \\ W_{16} = ROTL_1(W_{13} \oplus W_8 \oplus W_2 \oplus W_0) & i = 15 \end{cases}$$

These two keys satisfy $W_i^* = W_{i+1}$ for $0 \le i \le 78$.

Let $P = (A_0, B_0, C_0, D_0, E_0)$ and $P^* = (A_0^*, B_0^*, C_0^*, D_0^*, E_0^*)$ be two plaintexts encrypted under W and W^*, respectively. We denote the input to round i by $(A_i, B_i, C_i, D_i, E_i)$ (or with * when considering the encryption of P^* under W^*). If after the first round of the encryption of P under W the following holds:

$$A_1 = A_0^*; \quad B_1 = B_0^*; \quad C_1 = C_0^*; \quad D_1 = D_0^*; \quad E_1 = E_0^* \tag{1}$$

then this equality holds until round 20 of the encryption of P (or round 19 of the encryption of P^*). In order for the slid pair to retain its "slidness", the following equality must hold:

$$W_{20} + ROTL_5(A_{20}) + f_{20}(B_{20}, C_{20}, D_{20}) + E_{20} + K_{20} =$$

$$A_{21} \quad = \quad A_{20}^* =$$

$$W_{19}^* + ROTL_5(A_{19}^*) + f_{19}(B_{19}^*, C_{19}^*, D_{19}^*) + E_{19}^* + K_{19}$$

As $W_{20} = W_{19}^*$, $A_{20} = A_{19}^*$, $B_{20} = B_{19}^*$, $C_{20} = C_{19}^*$, $D_{20} = D_{19}^*$, $E_{20} = E_{19}^*$, the above holds whenever

$$f_{xor}(B_{20}, C_{20}, D_{20}) + K_{20} = f_{if}(B_{20}, C_{20}, D_{20}) + K_{19}. \tag{2}$$

For a random permutation, this equality holds with probability 2^{-32}. In the case of SHACAL-1, it was verified experimentally that the probability is close to 2^{-32} (see [10]).

If the transition between the IF rounds and the XOR rounds is successful, then the equality remains until round 40 of the encryption of P. Again, with probability close to 2^{-32} the transition in round 40 does not affect the equality between the respective intermediate encryption values. The same holds for the last transition in round 60.

Thus, Saarinen concluded that the probability that (P, P^*) is a slid pair assuming that it satisfies the condition of Equation (1) is 2^{-96}. To achieve such pairs, pairs of structures of 2^{32} chosen plaintexts are chosen, such that $SetP = (A, B, C, D, x)$ for some fixed A, B, C, D and all possible values of x, and $SetP^* = (y, A, ROTL_{30}(B), C, D)$ for all possible values of y. These structures ensure that for each $P \in SetP$ there exists $P^* \in SetP^*$ such that the pair

(P, P^*) satisfies the condition in Equation (1). Hence, 2^{64} pairs of structures are expected to contain a slid pair with a relatively high probability.

In order to detect the slid pairs, we note that for a slid pair we have

$$A_{80} = A^*_{79}; \quad B_{80} = B^*_{79}; \quad C_{80} = C^*_{79}; \quad D_{80} = D^*_{79}; \quad E_{80} = E^*_{79}, \qquad (3)$$

and this gives a 128-bit filtering on the ciphertexts that can be easily executed using a hash table. However, since we check a total number of 2^{128} pairs and our filtering is only on 128 bits, we expect that for a random permutation, one pair can also pass the filtering. In order to overcome this problem, we take 2^{65} pairs of structures, thus expecting to detect two slid pairs. Then we check whether the pairs suggest the same value for the subkey of the last round. If not, we collect some additional structures, find another slid pair and check again. With a high probability, the right subkey will be suggested at least twice, while for a random permutation the probability that a subkey is suggested twice is extremely low.

Therefore, the attack can distinguish between SHACAL-1 and a random permutation using about 2^{98} chosen plaintexts encrypted under two related keys, with time complexity of about 2^{98} encryptions.

We note that Saarinen has also proposed an algorithm that requires only 2^{32} operations to find such a slid pair in SHA-1. However, the algorithm assumes that the attacker can control the message block (i.e., the keys) directly, and thus it is not applicable to SHACAL-1 (unless it is in a chosen-key attack).

5 A Simple Related-Key Attack on SHACAL-1

The basic stage of our attack on SHACAL-1 is similar to the distinguishing attack presented by Saarinen [23]. We encrypt some pairs of structures under the two related keys and detect the candidate slid pairs. Now, each candidate slid pair suggests a value for two key words – the subkey used in the last round of the encryption under the key W^* and the subkey used in the first round of the encryption under W. At this stage, there is a difference between our attack and the attack in [23]: In order to reduce the data complexity of the attack we do not wait until the same key word is suggested twice, but rather continue with all the suggested subkey values. The wrong key values can be easily discarded in the last stage of our attack that will be described later, and hence we only need that the right key value will be amongst the suggested ones.

The algorithm of the basic stage of the attack is as follows:

1. Repeat the following M times, when M will be specified later:
   - Choose A, B, C, and D at random and ask for the encryption of $SetP = (A, B, C, D, x)$ for all possible values of x under W and of $SetP^* = (y, A, ROTL_{30}(B), C, D)$ for all possible values of y under W^*.
   - Search for candidate slid pairs, i.e., pairs of ciphertexts $T = (a, b, c, d, e) \in SetP$ and $T^* = (a^*, b^*, c^*, d^*, e^*) \in SetP^*$ such that $b^* = a, c^* = ROTL_{30}(b), d^* = c$, and $e^* = d$. Pass to the next stage all the candidate slid pairs, and the respective plaintext pairs denoted by $P = (A, B, C, D, X)$ and $P^* = (Y^*, A, ROTL_{30}(B), C, D)$, respectively.

2. For each candidate slid pair, compute W_0 and W_{80} using the formulas:

$$W_0 = Y^* - [ROTL_5(A) + f_{if}(B, C, D) + X + K_0] \quad (4)$$

$$W_{80} = a^* - [ROTL_5(a) + f_{xor}(b, c, d) + e + K_{79}] \quad (5)$$

3. Output all the pairs of values (W_0, W_{80}) suggested by candidate pairs.

After using the pair of related keys (W, W^*), we can repeat the attack with the related keys (W^*, W^{**}) that satisfy the relation

$$W_i^{**} = \begin{cases} W_{i+1}^* & 0 \le i \le 14 \\ W_{16}^* = ROTL_1(W_{13}^* \oplus W_8^* \oplus W_2^* \oplus W_0^*) & i = 15 \end{cases}$$

to retrieve two additional key words. This time the attack retrieves W_0^* and W_{80}^*, but due to the relation between W and W^*, these values are equal to W_1 and W_{81}, respectively. Then, the attack can be applied again. As all the keys are linearly dependent of each other, it is possible to combine the knowledge of each instance of the attack into a knowledge on the original key W. Therefore, if we repeat the attack 7 times, thus requiring 8 related keys, we get $64 \cdot 7 = 448$ linear equations in the bits of the key, and the rest of the key can be found by exhaustive search with time complexity of only 2^{64} encryptions. Note that if several candidates for the subkey values were suggested in some of the basic attacks, we perform the last stage of the attack for all the possible candidates, and still the time complexity is much below 2^{100} encryptions.

Now we want to compute the minimal possible value of M such that, with a high probability, in all the 7 applications of the basic stage of the attack the right key will be amongst the suggested ones. Using a Poisson approximation we get that if $M = 2^{64} \cdot t$ then the probability that in a single application of the basic attack no real slid pair will be found is e^{-t}. Hence, assuming that the basic stages are independent we get that the probability that in all the applications there will be at least one real slid pair is $(1 - e^{-t})^7$. For $t = 2^{1.5}$, we get $(1 - e^{-t})^7 \approx 0.65$, and hence the success probability of the attack is about 0.65. If we want a greater success probability, we can increase the data complexity. For example, for $t = 4$ the success probability is about 0.88 and for $t = 8$, the success probability is greater than 0.99.

Therefore, the total data complexity of the attack is $2^{1.5} \cdot 2^{64} \cdot 2^{33} \cdot 7 \approx 2^{101.3}$ related-key chosen plaintexts, and the time complexity is dominated solely by the encryption time.

The data complexity can be slightly reduced using an adaptive attack. We can ask for structures of plaintexts to be encrypted under two related keys, until two candidate slid pairs suggest the same subkeys (thus, with high probability, these are the right subkeys). Once this happens, there is no need to further encrypt plaintexts under these two related keys.

If only a smaller number of $k < 8$ related keys is available, we can perform the basic stage of the attack $k - 1$ times and find the rest of the key bits using exhaustive key search. We note that it is possible to reduce the time complexity

of this search by a factor of four, by considering the fact that for slid pairs there is a special relation between the intermediate encryption values in round 20 (presented in Equation (2)). If this relation is not satisfied, the key can be easily discarded without completing the full encryption.

Note that if the attack is performed less than seven times, we can reduce the number of chosen plaintexts and still expect that with a high probability, in all the applications of the basic attack there will be at least one slid pair. For example, for $k = 2$ we can take $t = 1$ and get success probability of 0.63, and for $k = 4$ we can take $t = 2$ and get success probability of 0.65.

The exact time complexity of the last stage depends on the number of subkey candidates suggested in the basic stages. Assuming that in each stage, besides the right subkey we encounter three wrong candidate values (that cannot be discarded) at most, the total time complexity of the attack is at most $2^{510+2(k-1)-64(k-1)} = 2^{510-62(k-1)}$ trial encryptions.

We conclude that our attack with k related keys has a data complexity of at most $(k-1) \cdot 2^{98.5}$ related-key chosen plaintexts, and a running time of at most $\max\{(k-1) \cdot 2^{98.5}, 2^{510-62(k-1)}\}$ trial encryptions.

6 Summary and Conclusions

In this paper we presented a simple related-key attack on SHACAL-1. The attack can be performed using between two and eight related-keys. The variant of the attack with eight related keys requires $2^{101.3}$ chosen plaintexts and has time complexity of $2^{101.3}$ encryptions. The attack is by far better than all the previously known related-key attacks on SHACAL-1.

Our results, following the results in [23], are based on the original variant of the *related-key attack* presented by Biham in 1993 [1].[3] Usually, slid pairs or their equivalent related-key counterparts can be found only if the round functions of the cipher are identical (for all the rounds). Hence, inserting a round constant to the round function of a block cipher seems to be sufficient to protect it with respect to related-key attacks.

In SHACAL-1, round constants are used in all the round functions but the fact that the constants are changed only once every 20 rounds and the fact that the round functions are also slightly changed in the same places can be used to construct the required pairs of plaintexts. Hence, the results of [23] and our attack show that related-key attacks can be mounted also on ciphers using round constants, if the constants are not chosen carefully.

We conclude that our key-recovery attack demonstrates, once again, that using a linear key schedule algorithm and relatively similar round functions is not a good way to design a secure block cipher.

[3] In 1996, Kelsey et al.[19] presented the *related-key differential attack*. This attack uses different ideas and the only similarity between it and Biham's attack is the fact that both attacks perform in the related-key model. The related-key rectangle technique, that was used in previous attacks on SHACAL-1, is an expansion of the related-key differential attack.

References

1. Eli Biham, *New Types of Cryptanalytic Attacks Using Related Keys*, Journal of Cryptology, Vol. 7, No. 4, pp. 229–246, Springer-Verlag, 1994.
2. Eli Biham, Rafi Chen, *Near-Collisions of SHA-0*, Advances in Cryptology, proceedings of CRYPTO 2004, Lecture Notes in Computer Science 3152, pp. 290–305, Springer-Verlag, 2004.
3. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, William Jalby, *Collisions of SHA-0 and Reduced SHA-1*, Advances in Cryptology, proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science 3621, pp. 36–57, 2005.
4. Eli Biham, Orr Dunkelman, Nathan Keller, *The Rectangle Attack – Rectangling the Serpent*, Advances in Cryptology, proceedings of EUROCRYPT '01, Lecture Notes in Computer Science 2045, pp. 340–357, Springer-Verlag, 2001.
5. Eli Biham, Orr Dunkelman, Nathan Keller, *New Results on Boomerang and Rectangle Attacks*, proceedings of Fast Software Encryption 9, Lecture Notes in Computer Science 2365, pp. 1–16, Springer-Verlag, 2002.
6. Eli Biham, Orr Dunkelman, Nathan Keller, *Rectangle Attacks on 49-Round SHACAL-1*, proceedings of Fast Software Encryption 10, Lecture Notes in Computer Science 2887, pp. 22–35, Springer-Verlag, 2003.
7. Eli Biham, Orr Dunkelman, Nathan Keller, *Related-Key Boomerang and Rectangle Attacks*, Advances in Cryptology, proceedings of EUROCRYPT '05, Lecture Notes in Computer Science 3494, pp. 507–525, Springer-Verlag, 2005.
8. Eli Biham, Orr Dunkelman, Nathan Keller, *Improved Slide Attacks*, private communication.
9. Eli Biham, Adi Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
10. Alex Biryukov, David Wagner, *Slide Attacks*, proceedings of Fast Software Encryption 6, Lecture Notes in Computer Science 1636, pp. 245–259, Springer-Verlag, 1999.
11. Alex Biryukov, David Wagner, *Advanced Slide Attacks*, Advances in Cryptology, proceedings of EUROCRYPT 2000, Lecture Notes in Computer Science 1807, pp. 586–606, Springer-Verlag, 2000.
12. Florent Chabaud, Antoine Joux, *Differential Collisions in SHA-0*, Advances in Cryptology, proceedings of CRYPTO 1998, Lecture Notes in Computer Science 1462, pp. 56–71, Springer-Verlag, 1998.
13. Orr Dunkelman, Nathan Keller, Jongsung Kim, *Related-Key Rectangle Attack on the Full SHACAL-1*, accepted to Selected Areas in Cryptography 2006, to appear in Lecture Notes in Computer Science.
14. Soichi Furuya, *Slide Attacks with a Known-Plaintext Cryptanalysis*, proceedings of Information and Communication Security 2001, Lecture Notes in Computer Science 2288, pp. 214–225, Springer-Verlag, 2002.
15. Helena Handschuh, Lars R. Knudsen, Matthew J. Robshaw, *Analysis of SHA-1 in Encryption Mode*, proceedings of CT-RSA 2001, Springer-Verlag Lecture Notes in Computer Science, vol. 2020, pp. 70–83, 2001.
16. Helena Handschuh, David Naccache, *SHACAL*, preproceedings of NESSIE first workshop, Leuven, 2000.
17. Seokhie Hong, Jongsung Kim, Guil Kim, Sangjin Lee, Bart Preneel, *Related-Key Rectangle Attacks on Reduced Versions of SHACAL-1 and AES-192*, proceedings of Fast Software Encryption 12, Lecture Notes in Computer Science 3557, pp. 368–383, Springer-Verlag, 2005.

18. John Kelsey, Tadayoshi Kohno, Bruce Schneier, *Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent*, proceedings of Fast Software Encryption 7, Lecture Notes in Computer Science 1978, pp. 75–93, Springer-Verlag, 2000.
19. John Kelsey, Bruce Schneier, David Wagner, *Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES*, Advances in Cryptology, proceedings of CRYPTO '96, Lecture Notes in Computer Science 1109, pp. 237–251, Springer-Verlag, 1996.
20. Jongsung Kim, Guil Kim, Seokhie Hong, Sangjin Lee, Dowon Hong, *The Related-Key Rectangle Attack — Application to SHACAL-1*, proceedings of ACISP 2004, Lecture Notes in Computer Science 3108, pp. 123–136, Springer-Verlag, 2004.
21. Jongsung Kim, Dukjae Moon, Wonil Lee, Seokhie Hong, Sangjin Lee, Seokwon Jung, *Amplified Boomerang Attack against Reduced-Round SHACAL*, Advances in Cryptology, proceedings of ASIACRYPT 2002, Lecture Notes in Computer Science 2501, pp. 243-253, Springer-Verlag, 2002.
22. Lars R. Knudsen, *Cryptanalysis of LOKI91*, proceedings of Auscrypt 1992, Lecture Notes in Computer Science 718, pp. 196–208, Springer-Verlag, 1993.
23. Markku-Juhani O. Saarinen, *Cryptanalysis of Block Ciphers Based on SHA-1 and MD5*, proceedings of Fast Software Encryption 10, Lecture Notes in Computer Science 2887, pp. 36–44, Springer-Verlag, 2003.
24. NESSIE – New European Schemes for Signatures, Integrity and Encryption. *http://www.nessie.eu.org/nessie*
25. NESSIE, *Portfolio of recommended cryptographic primitives.*
26. NESSIE, *Performance of Optimized Implementations of the NESSIE Primitives*, NES/DOC/TEC/WP6/D21/2.
27. US National Bureau of Standards, *Secure Hash Standard*, Federal Information Processing Standards Publications No. 180-2, 2002.
28. Eteinee Van Den Bogeart, Vincent Rijmen, *Differential Analysis of SHACAL*, NESSIE internal report NES/DOC/KUL/WP3/009/a, 2001.
29. David Wagner, *The Boomerang Attack*, proceedings of Fast Software Encryption 6, Lecture Notes in Computer Science 1636, pp. 156–170, 1999.
30. Xiaoyun Wang, Andrew C. Yao, Frances Yao, *Cryptanalysis on SHA-1*, Cryptographic Hash Workshop, NIST, Gaithersburg, 2005.
31. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, Xiuyuan Yu, *Cryptanalysis of the Hash Functions MD4 and RIPEMD*, Advances in Cryptology, proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science 3494, pp. 1–18, 2005.
32. Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu, *Finding Collisions in the Full SHA-1*, Advances in Cryptology, proceedings of CRYPTO 2005, Lecture Notes in Computer Science 3621, pp. 17–36, 2005.
33. Xiaoyun Wang, Hongbo Yu, *How to Break MD5 and Other Hash Functions*, Advances in Cryptology, proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science 3494, pp. 19–35, 2005.
34. Xiaoyun Wang, Hongbo Yu, Yiqun Lisa Yin, *Efficient Collision Search Attacks on SHA-0*, Advances in Cryptology, proceedings of CRYPTO 2005, Lecture Notes in Computer Science 3621, pp. 1–16, 2005.

Impossibility Proofs for RSA Signatures in the Standard Model

Pascal Paillier

Cryptography & Innovation, Security Labs, Gemalto
pascal.paillier@gemalto.com

Abstract. It is well-known that RSA signatures such as FDH, PSS or PSS-R are as secure as RSA is hard to invert in the random oracle (RO) model. Such proofs, however, have never been discovered in the standard model. This paper provides an explanation of this gap by pointing out a strong *impossibility* of equivalence between inverting RSA and any form of unforgeability for a wide class of RSA signatures. In particular, our impossibility results explicitly assume that the public key is made of a single RSA instance, that hash functions involved in the signature padding are unkeyed and that key generation fulfils a natural property which we call instance-non-malleability. Beyond showing that any RSA-based signature scheme of that type black-box separates the RO model from the standard model in a strong sense, our work leaves the real-life security of well-known signatures in a state of uncertainty.

1 Introduction

Background. Provable security is a relatively young field in cryptography. Historically, the security of cryptosystems has been a most central subject, but the study of security reductions relating schemes to computational problems of reference has been scattered until fairly recently. In little over a decade, the subject has attracted enormous interest and accumulated an impressive body of results. Quite surprisingly, years of active research on RSA signatures such as Full Domain Hash [3,7] (FDH) or Probabilistic Signature Scheme [17] (PSS/PSS-R) have brought no evidence that the standard-model security of these is satisfactory. The question of coming up with a security proof for these schemes remains one of the most challenging open problems of modern cryptography. The practical implications of such a proof are of prime importance to the security industry due to the fact that most cryptographic standards [17,12] for digital signatures that are in use today rely on RSA-based constructions.

Proofs involving idealized versions of the hash functions, pseudo-random functions or blockciphers used within the signature padding procedure have progressively emerged over the years. The random oracle (RO) model [3], despite recent critics about its limitations [1,6,13], has long served cryptographers in their work of design and assessment. Whenever a proof that a signature scheme reaches strong unforgeability is found in the random oracle model, it has been a

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 31–48, 2007.

tradition to extend its scope to the standard model thanks to the heuristic argument (often referred to as the RO model heuristic) that a RO-model security proof somewhat guarantees the security of standard-model instantiations of that signature scheme under the same complexity assumptions.

This paper provides evidence that this heuristic argument might not be well-founded for real-life RSA signatures *i.e.,* those defined in current cryptographic standards[1]. Real-life RSA signatures are basic: they employ paddings based on unkeyed hash functions and have a public key consisting of only one RSA instance. What we show is that breaking *any* form of unforgeability w.r.t. *any* signature scheme of this type *cannot* be equivalent to inverting RSA in the standard model. Our impossibility results are partial in the sense that we disprove equivalence under the assumption that the key generation is *instance-non-malleable.* In a nutshell, this property means that extracting e-th roots modulo n is not any easier if one is given an e'-th root extractor modulo n' for $(n', e') \neq (n, e)$. Although scarcely studied so far, it seems reasonable to conjecture that commonly used RSA instance generators are in turn instance-non-malleable, unless voluntarily constructed otherwise [21].

Our results point out that real-life RSA signatures just cannot reach a maximal security level *i.e.,* equivalence to inverting RSA, or rather, that if one could prove in the future that they can, that proof would force the distribution of public keys to realize some form of instance-malleability. Subsequently, our work provides a strong black-box separation between the standard and the RO model in which it is well-known that forging *e.g.* PSS signatures is equivalent to inverting RSA. Previous works have observed similar separations in specific contexts.

Prior Work. The present work is related to several important results showing that a number of schemes proven secure in the RO model cannot be securely instantiated in the standard model. Canetti, Goldreich and Halevi [6] suggested artificial yet concrete examples of signature and encryption schemes featuring this property. Bellare, Boldyreva and Palacio [1] showed a simple hash variant of ElGamal which is CCA-preserving in the RO model but does not realize this property in the standard model. Goldwasser and Tauman have reported the existence of a secure 3-round ID protocol that cannot be Fiat-Shamir-converted into a secure signature scheme in the standard model. In another direction, Boldyreva and Fischlin [4] recently considered the question of instantiating random oracles (*e.g.* in FDH) with realizable functions such as verifiable pseudorandom functions (VRFs). They showed in particular that VRFs cannot generically instantiate random oracles. Another negative and wider result on instantiations of FDH was recently found by Dodis, Oliveira and Pietrzak [8]. More recently, Paillier and Vergnaud [15] discovered impossibility results for Fiat-Shamir transformed [10] and assimilated signatures such as DSA, ElGamal and Schnorr signatures. Even more recently, Paillier and Villar [16] give a reformulation of a long-lived folklore impossibility result due to Williams [18,20] (see also [11]) to show that factoring-based encryption schemes using a single modulus as public key cannot

[1] Formally refuting the RO-model heuristic here would require a polynomial attack on RSA signatures. Instead, we show that nothing prevents the *existence* of such attacks.

be CCA secure under the factoring assumption, using the following two-step reasoning. First, they formally reformulate Williams' observation that no *key-preserving* security reduction exists which reduces factoring to CCA security. Those reductions are limited to make queries to the adversarial oracle that involve the very same public key as the one they are given as input. Second, they extend the impossibility to arbitrary reductions under the assumption that the key generation is non-malleable or in other words, that non-key-preserving oracle calls bring essentially no computational advantage to security reductions.

This paper extends the results of [16] to the case of RSA signatures. We follow the same two-stage approach. Based on a refinement of non-malleability more adapted to RSA key generators which we call instance-non-malleability, we closely relate the existence of unrestricted security reductions to the one of key-preserving reductions, which are easily shown not to exist using a diagonal argument as in [16, Corollary 1]. We emphasize that impossibility results in the key-preserving case alone are in fact very simple and lead to one-line proofs that are folklore since the late eighties. It seems to us, however, that introducing instance-non-malleability is the only way to formally extend the non-existence of key-preserving reductions to the non-existence of arbitrary reductions, which definitely leads to a much stronger statement. We thus view our contributions as a natural extension of the impossibility results originally pointed out by Williams.

This paper focuses on the case of RSA signatures although similar impossibility results are applicable to RSA encryption such as RSA-OAEP. Independently from this work, Brown [5] recently restated the (folklore) non-existence of key-preserving security reductions (under the name of simple reductions) in the case of RSA encryption, that would yield a key-preserving equivalence between IND-CCA security and inverting RSA. We finally comment that including an additional random string in the public key (*i.e.,* using at least one family of hash functions in the message padding) or allowing multiple RSA instances in the public key as in [11,9] is enough to avoid our impossibility results completely[2]. We therefore preconize the inclusion of an additional random string in the public key as a simple fix in practical implementations of RSA signatures and encryption. We refer to the full version of this work [14] for further comments on these countermeasures as well as an explanation as to why the security proofs we consider here and claim not to exist in the real world may be sound in the RO model.

Our contributions. We focus on several security notions for an arbitrary real-life RSA-based signature scheme and assuming a black-box equivalence with inverting RSA in the standard model, we analyze the effect that such an equivalence exerts on the security profile of the scheme *i.e.,* the map of all known security reductions related to this scheme. We start by considering universal unforgeability under key-only attack[3] and find that an equivalence with RSA nullifies resistance against chosen message attacks. This result is in itself sufficient to prove that

[2] Goldwasser-Micali-Rivest and Dwork-Naor constructions precisely do this and are indeed shown to be equivalent to RSA.

[3] Also known as no-message attack.

there exists no RSA-based scheme which is chosen-message unforgeable under the RSA assumption, unless inverting RSA is easy or RSA is instance-malleable. We further extend this result to existential unforgeability under key-only attack and show that an equivalence with RSA nullifies resistance against known message attacks. Investigating stronger assumptions, we then proceed to show that there is no hope one can ever base the unforgeability of RSA signatures on the One-More RSA assumption [2] or the gap RSA assumption *i.e.,* the computational gap between inverting RSA and factoring.

Our proof technique is based on the construction of black-box meta-reductions as initiated in [7,15,16]. This increasingly popular technique yields simple impossibility proofs featuring a perfect preservation of success probabilities. Combining it with instance-non-malleability, unfortunately, does not allow to preserve the constructive aspect of reductionist proofs based on meta-reductions[4]. Finally, we emphasize that the goal of this paper is to disprove that common RSA signatures are maximally secure and that our results are essentially of theoretical interest. We do not report weaknesses or attacks on either of the signature schemes we consider here, and in particular we do not deny that forging RSA signatures is likely to be intractable in practice.

Roadmap. Section 2 provides a number of definitional facts about instance generators, RSA signatures and related security notions. We introduce instance-malleability in Section 3 and establish a crucial property of instance-non-malleable generators in terms of security games. We proceed to show in Section 4.1 that an equivalence between inverting RSA and universal forgeries under key-only attacks imply the existence of efficient chosen message attacks. Section 4.2 further extends this result to existential forgeries. We introduce root extraction attacks in Section 5 and use these to show why RSA signatures cannot be proven unforgeable under the One-More RSA assumption either. We conclude in Section 6.

2 Preliminary Facts

2.1 Black-Box Reductions

We adopt the same notations as in [16]. A black-box[5] reduction $\mathcal{R}$ between two computational problems P_1 and P_2 is a probabilistic algorithm $\mathcal{R}$ which solves P_1 given black-box access to an oracle solving P_2. We write $P_1 \Leftarrow_{\mathcal{R}} P_2$ when $\mathcal{R}$ is known to reduce P_1 to P_2 in polynomial extra time (in a security parameter), counting oracle calls for one elementary step. Note that $\mathcal{R}$ can be polynomial even when no polynomial-time algorithm solving P_2 is known to exist. $P_1 \Leftarrow P_2$ states that some polynomial $\mathcal{R}$ exists such that $P_1 \Leftarrow_{\mathcal{R}} P_2$ and $P_1 \equiv P_2$ means $P_1 \Leftarrow P_2$ and $P_2 \Leftarrow P_1$. $\mathsf{Succ}\,(P, \tau)$ denotes the maximal success probability taken over all probabilistic algorithms solving P in no more than τ elementary steps.

[4] See [14] for further details.

[5] All reductions considered in this paper are fully black-box.

Similarly, $\mathsf{Succ}\left(P_1 \Leftarrow P_2, \varepsilon, \tau, \ell\right)$ is the maximal success probability of algorithms solving P_1 in no more than τ steps and at most ℓ calls to an oracle solving P_2 with probability ε.

2.2 RSA and Related Computational Problems

Root Extraction. Solving the root extraction problem consists in computing $\mathsf{RSA}(n, e, y) = y^{1/e} \bmod n$ given random integers n, e such that $\gcd(e, \phi(n)) = 1$ where $\phi(n) = |\mathbb{Z}_n^*|$ and $y \leftarrow \mathbb{Z}_n$. Because of its random self-reducibility [19], the hardness of computing $\mathsf{RSA}(n, e, y)$ is essentially independent from the choice of y and rather depends on n and e. We denote $\mathsf{RSA}(n, e, \cdot)$ the problem of computing e-th roots modulo n. Since the intractability of $\mathsf{RSA}(n, e, \cdot)$ is variable and strongly related to the form of n, cryptographic applications only rely on hard instances by defining some instance generator $\mathcal{RSA}$. Given a security parameter k, $\mathcal{RSA}(1^k)$ generates a hard instance (n, e), as well as the side information $d = e^{-1} \bmod \phi(n)$. Abusing notations, we will indifferently write $(n, e, d) \leftarrow \mathcal{RSA}(1^k)$ or $(n, e) \leftarrow \mathcal{RSA}(1^k)$ to denote a random selection of an RSA key, d being output or not depending on the context. A probabilistic algorithm $\mathcal{A}$ is said to (ε, τ)-invert $\mathcal{RSA}$ or equivalently (ε, τ)-break $\mathsf{INV}\left[\mathcal{RSA}\right]$ when $\Pr\left[(n, e, d) \leftarrow \mathcal{RSA}(1^k), y \leftarrow \mathbb{Z}_n \ : \ \mathcal{A}(n, e, y) = y^d \bmod n\right] \geq \varepsilon$ where the probability is taken over the coin flips of $\mathcal{A}$ and $\mathcal{RSA}$ and $\mathcal{A}$ stops after τ steps. The RSA assumption (where the instance generator $\mathcal{RSA}$ is often referred to implicitly) states that $\mathsf{Succ}\left(\mathsf{INV}\left[\mathcal{RSA}\right], \tau\right) = \mathsf{negl}\left(k\right)$ for $\tau = \mathsf{poly}\left(k\right)$.

The One-More RSA Problem [2]. $\mathsf{INV}\left[\mathcal{RSA}\right]$ is generalized to a family of computational problems $\ell\text{-}\mathsf{OM}\left[\mathcal{RSA}\right]$ for $\ell \geq 0$. A probabilistic algorithm $\mathcal{A}$ breaks $\ell\text{-}\mathsf{OM}\left[\mathcal{RSA}\right]$ when given a random instance $(n, e) \leftarrow \mathcal{RSA}(1^k)$, $\ell + 1$ random integers $y_0, y_1, \ldots, y_\ell \leftarrow \mathbb{Z}_n$ and oracle access to $\mathsf{RSA}(n, e, \cdot)$, $\mathcal{A}$ outputs the e-th roots of $y_0, \ldots, y_\ell$ in no more than ℓ calls to the oracle. Formally, $\mathcal{A}$ is said to (ε, τ)-break $\ell\text{-}\mathsf{OM}\left[\mathcal{RSA}\right]$ when

$$\Pr\left[\begin{array}{l}(n, e, d) \leftarrow \mathcal{RSA}(1^k), \\ y_0, y_1, \ldots, y_\ell \leftarrow \mathbb{Z}_n\end{array} \ : \ \mathcal{A}^{\mathsf{RSA}(n,e,\cdot)}(n, e, y_0, \ldots, y_\ell) = (y_0^d, \ldots, y_\ell^d)\right] \geq \varepsilon \ ,$$

where the outputs are modulo n, the probability is taken over the random tapes of $\mathcal{A}$ and of the experiment, the runtime of $\mathcal{A}$ is upper-bounded by τ and $\mathcal{A}$ sends at most ℓ requests to $\mathsf{RSA}(n, e, \cdot)$. Note that $0\text{-}\mathsf{OM}\left[\mathcal{RSA}\right]$ is identical to $\mathsf{INV}\left[\mathcal{RSA}\right]$ by definition. Also, $\ell_2\text{-}\mathsf{OM}\left[\mathcal{RSA}\right] \Leftarrow \ell_1\text{-}\mathsf{OM}\left[\mathcal{RSA}\right]$ if $\ell_2 \geq \ell_1$. The One-More RSA assumption states that $\mathsf{Succ}\left(\ell\text{-}\mathsf{OM}\left[\mathcal{RSA}\right], \tau\right) = \mathsf{negl}\left(k\right)$ for $\tau = \mathsf{poly}\left(k\right)$ and $\ell = \mathsf{poly}\left(k\right)$.

Total Break, Factoring and Relations among Problems. A total break of $\mathcal{RSA}$ states that computing d from (n, e) is not hard in average *i.e.*, when $(n, e, d) \leftarrow \mathcal{RSA}(1^k)$. A probabilistic algorithm $\mathcal{A}$ (ε, τ)-breaks $\mathsf{FACT}\left[\mathcal{RSA}\right]$ when

$$\Pr\left[(n, e, d) \leftarrow \mathcal{RSA}(1^k) \ : \ \mathcal{A}(n, e) = d\right] \geq \varepsilon$$

taken over the random coins of $\mathcal{A}$ and $\mathcal{RSA}$ and $\mathcal{A}$ runs in time at most τ. Also, we define $\mathsf{GAP}[\mathcal{RSA}]$ as the problem of computing d from $(n, e) \leftarrow \mathcal{RSA}(1^k)$ given oracle access to $\mathsf{RSA}(n, e, \cdot)$ *i.e.,* $\mathcal{A}$ (ε, τ)-breaks $\mathsf{GAP}[\mathcal{RSA}]$ when

$$\Pr\left[(n, e, d) \leftarrow \mathcal{RSA}(1^k) \ : \ \mathcal{A}^{\mathsf{RSA}(n,e,\cdot)}(n, e) = d\right] \geq \varepsilon \ .$$

It is easily seen that for any instance generator $\mathcal{RSA}$, one has

$$\begin{array}{ccc} \mathsf{GAP}[\mathcal{RSA}] & \Leftarrow & \mathsf{FACT}[\mathcal{RSA}] \\ \Downarrow & & \Downarrow \\ \mathsf{OM}[\mathcal{RSA}] & \Leftarrow & \mathsf{INV}[\mathcal{RSA}] \end{array}$$

2.3 Real-Life RSA Signatures

Standardized RSA signatures [17,12] use simple padding functions mostly composed of cascaded or interleaved unkeyed hash functions, exclusive-ors and concatenations of bitstrings with fixed patterns. We therefore define RSA signatures in a way that reflects this feature. A real-life RSA-based signature scheme $\mathcal{S}$ with security parameter k is described as the combination of an RSA instance generator $\mathcal{RSA}$ with a padding (or redundancy) function

$$\mu : \{0,1\}^{\mathsf{m}} \times \{0,1\}^{\mathsf{r}} \rightarrow \{0,1\}^{\mathsf{w}}$$

where the descriptions of μ, of $\mathsf{m}, \mathsf{w} \geq 1$ and of $\mathsf{r} \geq 0$ are functions of k. It may be the case that $\mathsf{m} = *$ if signing messages of arbitrary length is supported. We impose that μ be *verifiable*: there must be a function ν such that for any $m \in \{0,1\}^{\mathsf{m}}, y \in \{0,1\}^{\mathsf{w}}$, $\nu(m, y) = 1$ if there exists $r \in \{0,1\}^{\mathsf{r}}$ such that $y = \mu(m, r)$ and 0 otherwise. The primary role of a padding function is to destroy the homomorphic property of modular exponentiation. To this end, μ generally involves one or several hash functions such as SHA-1. A secondary but important feature is to make signatures probabilistic, in which case $\mathsf{r} > 0$. We identify $\mathcal{S} = (\mathcal{RSA}, \mu)$ to a tuple of probabilistic algorithms Gen, Sign and Ver defined as follows.

Key generation. $\textsc{Gen}(1^k)$ runs $\mathcal{RSA}(1^k)$ to get (n, e, d). The secret key is (n, d) while the public key is (n, e).

Sign. Given a secret key (n, d) and a message $m \in \{0,1\}^{\mathsf{m}}$, $\textsc{Sign}(n, d, m)$ randomly selects $r \leftarrow \{0,1\}^{\mathsf{r}}$ and computes the signature $s = \mu(m, r)^d \bmod n$.

Verify. $\textsc{Ver}(n, e, m, s)$ returns $\nu(m, s^e \bmod n)$.

We impose that μ and ν are both efficiently computable *i.e.,* in time $\mathsf{poly}(k)$. Note that the domain and range of μ are independent from the public key but that the number of output bits w depends on the key length. Although easily extendable, this setting captures RSA signature schemes described in industrial standards such as PSS. Through the rest of the paper, $\mathcal{S}$ stands for a real-life RSA signature scheme and $\mathcal{S}_{n,e}(m)$ denotes the set of all possible signatures on m with respect to $\mathcal{S}$ and public key (n, e).

2.4 Security Notions for Real-Life RSA Signatures

Security notions combine an adversarial goal with an attack model. The attacker is seen as a probabilistic polynomial time algorithm that attempts to fulfill its goal while being given a number of computational resources. The attacker may interact with the scheme in different ways.

Adversarial goals. We say that a signature scheme is *breakable* (BK) when an adversary extracts the secret key matching a randomly chosen public key $(n, e) \leftarrow \mathcal{RSA}(1^k)$. The scheme is said to be *universally forgeable* (UF) when there exists an adversary that returns a valid signature on a randomly chosen message $m \leftarrow \{0,1\}^{\mathsf{m}}$ given as input. The notion of *existential forgeability* (EF) is similar but allows the adversary to choose freely the value of the signed message. These notions are classical [11]. Since the focus is on RSA signatures, we introduce a specific adversarial goal according to which the adversary attempts to extract the e-th root of a randomly chosen element $y \leftarrow \mathbb{Z}_n$ for a randomly chosen key $(n, e) \leftarrow \mathcal{RSA}(1^k)$. We say that the scheme is *root-extractable* (RE) if this goal can be fulfilled in probabilistic polynomial time. It is easily seen that this goal is weaker than BK but stronger than UF.

Attack models. We consider several attack scenarios in this paper. In a *key-only attack* (KOA), the adversary is given nothing else than a public key as input. A *known message attack* (KMA) consists in giving as input to the attacker a list $(m_1, s_1), \ldots, (m_\ell, s_\ell)$ of pairwise distinct message-signature pairs. In a *chosen message attack* (CMA), the adversary is given adaptive access to a signing oracle.

Relations among security levels. As in [15,16], we view security notions as computational problems *e.g.* $\mathsf{UF\text{-}KMA}[\mathcal{S}]$ is the problem of computing a universal forgery under known message attack. This notation allows to relate security notions using reductions. In the case of KMA or CMA, we denote by $\ell\text{-}\mathsf{GOAL\text{-}ATK}[\mathcal{S}]$ the problem of breaking GOAL in no more than ℓ calls to the resource defined by ATK. Thus, breaking $\ell\text{-}\mathsf{EF\text{-}CMA}[\mathcal{S}]$ authorizes at most ℓ calls to the signing oracle to break EF. We recall that $\mathsf{GOAL\text{-}CMA}[\mathcal{S}] \Leftarrow \mathsf{GOAL\text{-}KMA}[\mathcal{S}] \Leftarrow \mathsf{GOAL\text{-}KOA}[\mathcal{S}]$ for any RSA signature scheme $\mathcal{S}$ and adversarial goal $\mathsf{GOAL} \in \{\mathsf{BK}, \mathsf{RE}, \mathsf{UF}, \mathsf{EF}\}$. Fig. 1 displays the map of black-box reductions among security levels that will be of interest for this work.

3 Instance-Malleability of RSA Instance Generators

We refine the definition of non-malleability suggested by [16] into one that is better suited to the context of RSA key generators.

3.1 What Is Instance-Malleability?

Interpretation. We say that $\mathcal{RSA}$ is instance-malleable if extracting roots with respect to a randomly selected instance (n, e) is easier when given repeated access to an oracle that extracts e'-th roots modulo n' for other instances $(n', e') \neq$

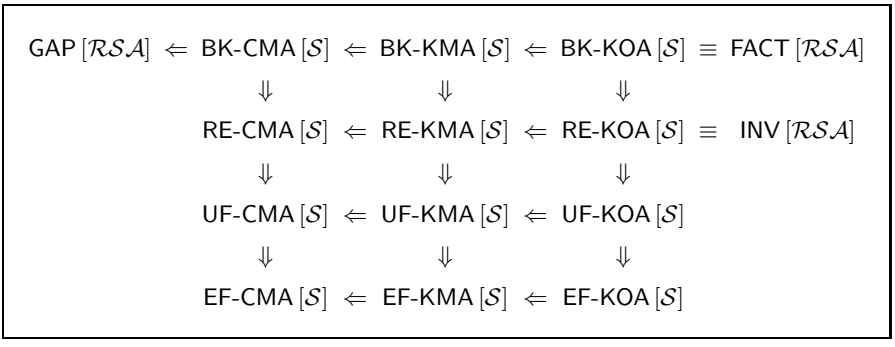

Fig. 1. Relations among security notions for real-life RSA signatures

(n, e) in the range of $\mathcal{RSA}(1^k)$. A typical example of instance-malleability is when public exponents generated by $\mathcal{RSA}$ are easy-to-factor integers. Given $(n, e = e_1 e_2)$, e-th roots modulo n are easy to compute when e_1-th and e_2-th root extractions modulo n are provided. Another example is when the distribution of n induced by $\mathcal{RSA}(1^k)$ contains moduli of variable bitlength and number of factors. It is indeed trivial to compute e-th roots modulo n given an e-th root extractor modulo $n' = \alpha n$ if it is the case that both (n, e) and (n', e) are proper outputs[6] of $\mathcal{RSA}(1^k)$. We observe that common RSA-based cryptosystems rely on generators which precisely seem to avoid any form of malleability by construction: it is generally the case that the bitlength of n is fixed to k, that the number of prime factors is equal to two and that e is limited to be a prime number or a constant value. The simple malleability properties discussed above do not apply then and it seems reasonable to believe that instance-non-malleability is de facto realized in a strong sense. We now give a proper definition.

Formal Definition. Following the methodology of [16], we define two games in which a probabilistic algorithm $\mathcal{R}$ attempts to extract an e-th root modulo n where $(n, e) \leftarrow \mathcal{RSA}(1^k)$. We assume that $\mathcal{R}$ has access to some perfect oracle $\mathcal{A}(n, e, \mathsf{aux})$ implicitly parameterized by the very same instance (n, e). Here aux denotes an auxiliary input depending on how $\mathcal{A}$ is specified and $\mathcal{R}$ may submit any admissible value for aux to the oracle. Since $\mathcal{A}$ is perfect *i.e.*, has success probability equal to one, $\mathcal{A}$ can be identified to the problem it solves which we denote also by $\mathcal{A}$. We impose $\mathcal{A}$ to be *trivially reducible* to $\mathsf{INV}[\mathcal{RSA}]$, meaning that there must exist a reduction algorithm $\mathcal{T}$ such that for any $(n, e) \in \mathsf{Range}\left(\mathcal{RSA}(1^k)\right)$ and any auxiliary input aux, $\mathcal{A}(n, e, \mathsf{aux})$ can be solved with probability one with a single call to $\mathsf{RSA}(n, e, \cdot)$ in time $t_{\mathcal{A}} = \mathsf{poly}(k)$. Game 0 is the game as per the above wherein algorithm $\mathcal{R}$ has success probability

$$\mathsf{Succ}^0(\mathcal{R}, \mathcal{A}, \tau, \ell) = \Pr\left[\begin{array}{c}(n, e, d) \leftarrow \mathcal{RSA}(1^k) \\ y \leftarrow \mathbb{Z}_n\end{array} : \mathcal{R}^{\mathcal{A}(n,e,\cdot)}(n, e, y) = y^d \bmod n\right]$$

[6] This requirement has to hold given current security notions, see [14].

where $\mathcal{R}$ runs in (extra) time at most τ and makes at most ℓ queries to $\mathcal{A}(n, e, \cdot)$. The success probability of Game 0 is then defined as

$$\mathsf{Succ}^0\left(\mathcal{A}, \tau, \ell\right) = \max_{\mathcal{R}} \mathsf{Succ}^0\left(\mathcal{R}, \mathcal{A}, \tau, \ell\right)$$

where the maximum is taken over all probabilistic algorithms $\mathcal{R}$ conforming to Game 0. In Game 1, the challenged algorithm is also given access to an oracle $\mathsf{RSA}(n', e', y)$ which returns $y^{1/e'} \bmod n'$ with probability one for any $(n', e') \neq (n, e)$ provided that $(n', e') \in \mathsf{Range}\left(\mathcal{RSA}(1^k)\right)$ and $y \in \mathbb{Z}_{n'}$. Its success probability $\mathsf{Succ}^1\left(\mathcal{R}, \mathcal{A}, \tau, \ell\right)$ is then

$$\Pr\left[\begin{matrix}(n, e, d) \leftarrow \mathcal{RSA}(1^k) \\ y \leftarrow \mathbb{Z}_n\end{matrix} \ : \ \mathcal{R}^{\mathcal{A}(n,e,\cdot),\mathsf{RSA}(\cdot,\cdot,\cdot)}(n, e, y) = y^d \bmod n\right] ,$$

where $\mathcal{R}$ runs in time at most τ, makes $\ell_{\mathcal{A}}$ queries to $\mathcal{A}(n, e, \cdot)$ and ℓ_{RSA} queries of the form (n', e', y) to $\mathsf{RSA}(\cdot, \cdot, \cdot)$ and $\ell_{\mathcal{A}} + \ell_{\mathsf{RSA}} \leq \ell$. Letting

$$\mathsf{Succ}^1\left(\mathcal{A}, \tau, \ell\right) = \max_{\mathcal{R}} \mathsf{Succ}^1\left(\mathcal{R}, \mathcal{A}, \tau, \ell\right) \ ,$$

the maximum being taken over all probabilistic algorithms $\mathcal{R}$ playing Game 1, we define

$$\Delta^{\mathsf{INV}}_{\mathcal{RSA}}\left(\tau, \ell\right) = \max_{\mathcal{A} \Leftarrow \mathsf{INV}[\mathcal{RSA}]} \left| \mathsf{Succ}^1\left(\mathcal{A}, \tau, \ell\right) - \mathsf{Succ}^0\left(\mathcal{A}, \tau, \ell\right) \right|$$

where the maximum is taken over all computational problems $\mathcal{A}$ trivially reducible to $\mathsf{INV}\left[\mathcal{RSA}\right]$ in polynomial time.

Definition 1 (Instance-Non-Malleability). *Note that* $\Delta^{\mathsf{INV}}_{\mathcal{RSA}}\left(\tau, 0\right) = 0$ *for any integer* τ*. An instance generator* $\mathcal{RSA}$ *is said to be instance-non-malleable when* $\Delta^{\mathsf{INV}}_{\mathcal{RSA}}\left(\tau, \ell\right) = \mathsf{negl}\left(k\right)$ *when* $\tau = \mathsf{poly}\left(k\right)$ *and* $\ell = \mathsf{poly}\left(k\right)$*.*

3.2 Application to Security Games

The property of instance-non-malleability allows one to relate reduction algorithms $\mathcal{R}$ such that $\mathsf{INV}\left[\mathcal{RSA}\right] \Leftarrow_{\mathcal{R}} \mathcal{A}$ to algorithms solving the above games, provided that $\mathcal{A}$ is trivially reducible to $\mathsf{INV}\left[\mathcal{RSA}\right]$.

Lemma 1. *Let* $\mathcal{A}$ *be a computational problem trivially reducible to* $\mathsf{INV}\left[\mathcal{RSA}\right]$ *in time* $t_{\mathcal{A}}$*. Then for any positive integers* τ, ℓ *and any* $\varepsilon \in (0, 1)$*,*

$$\begin{aligned}\mathsf{Succ}\left(\mathsf{INV}\left[\mathcal{RSA}\right] \Leftarrow \mathcal{A}, \varepsilon, \tau, \ell\right) &\leq \mathsf{Succ}^1\left(\mathcal{A}, \tau + \ell \cdot t_{\mathcal{A}}, \ell\right) \\ &\leq \mathsf{Succ}^0\left(\mathcal{A}, \tau + \ell \cdot t_{\mathcal{A}}, \ell\right) + \Delta^{\mathsf{INV}}_{\mathcal{RSA}}\left(\tau + \ell \cdot t_{\mathcal{A}}, \ell\right) \ .\end{aligned}$$

Proof. (First $\leq$) Assume there exists a probabilistic algorithm $\mathcal{R}$ such that $\mathsf{INV}\left[\mathcal{RSA}\right] \Leftarrow_{\mathcal{R}} \mathcal{A}$ which succeeds with probability $\varepsilon_{\mathcal{R}}$ in time τ by making at most ℓ queries to an oracle $\mathcal{A}_{\mathcal{R}}$ solving $\mathcal{A}$ with probability ε. We build an algorithm $\mathcal{M}$ which conforms to Game 1 and succeeds with identical probability

and running time as above. Algorithm $\mathcal{M}$ makes use of the trivial reduction $\mathcal{T}$ as a subroutine and behaves as follows. Given an RSA instance $(n, e) \leftarrow \mathcal{RSA}(1^k)$ and $y \leftarrow \mathbb{Z}_n$, $\mathcal{M}$ runs $\mathcal{R}$ over (n, e, y). Now for each oracle query $\mathcal{A}_{\mathcal{R}}(n_i, e_i, \mathsf{aux}_i)$ that $\mathcal{R}$ makes to $\mathcal{A}_{\mathcal{R}}$, $\mathcal{M}$ either runs $\mathcal{A}(n, e, \mathsf{aux}_i)$ if $(n_i, e_i) = (n, e)$ and forwards the output to $\mathcal{R}$, or runs $\mathcal{T}(n_i, e_i, \mathsf{aux}_i)$ if $(n_i, e_i) \neq (n, e)$, supplying $\mathcal{T}$ with oracle access to $\mathsf{RSA}(n_i, e_i, \cdot)$, and forwards the result to $\mathcal{R}$. $\mathcal{M}$ eventually returns the output of $\mathcal{R}$. Since $\mathcal{T}$ succeeds with probability one, the simulation of $\mathcal{A}_{\mathcal{R}}$ is perfect for any $\varepsilon \in (0, 1)$. $\mathcal{M}$ requires extra time at most $\ell \cdot t_{\mathcal{A}}$ and totalizes at most ℓ calls to $\mathcal{A}(n, e, \cdot)$ and $\mathsf{RSA}(\cdot, \cdot, \cdot)$ with instances $(n', e') \neq (n, e)$. (Second $\leq$) This follows from the definition of $\Delta^{\mathsf{INV}}_{\mathcal{RSA}}(\tau, \ell)$. □

4 Impossibility of Equivalence with Inverting RSA

4.1 Universal Unforgeability and Chosen-Message Security

We focus on the standard-model security level $\mathsf{UF\text{-}KOA}[\mathcal{S}]$ of a real-life RSA-based signature scheme $\mathcal{S}$ combining an instance-non-malleable instance generator $\mathcal{RSA}$ and a padding function μ.

Theorem 1. *Let $\mathcal{S} = (\mathcal{RSA}, \mu)$ be an RSA signature scheme and assume that $\mathcal{RSA}$ is instance-non-malleable. If $\mathsf{UF\text{-}KOA}[\mathcal{S}]$ is equivalent to $\mathsf{INV}[\mathcal{RSA}]$ then $\mathsf{RE\text{-}CMA}[\mathcal{S}]$ is polynomial.*

Proof. The proof comes in two stages. In the first stage, we focus on Game 0 with respect to $\mathcal{A} \triangleq \mathsf{UF\text{-}KOA}[\mathcal{S}]$ and use a reduction algorithm successfully playing Game 0 to break $\mathsf{RE\text{-}CMA}[\mathcal{S}]$. We then make use of instance-non-malleability and Lemma 1 to conclude in stage 2.

Stage 1. Let us assume that there exists a reduction algorithm $\mathcal{R}$ as per Game 0 which $(\varepsilon_{\mathcal{R}}, \tau)$-inverts $\mathcal{RSA}$ given oracle access to a perfect oracle $\mathcal{A}$ that breaks $\mathsf{UF\text{-}KOA}[\mathcal{S}]$ and let ℓ be the maximal number of times $\mathcal{R}$ runs $\mathcal{A}$. We build an algorithm $\mathcal{M}$ which $(\varepsilon_{\mathcal{M}}, \tau_{\mathcal{M}})$-breaks ℓ-$\mathsf{RE\text{-}CMA}[\mathcal{S}]$ with $\varepsilon_{\mathcal{M}} \geq \varepsilon_{\mathcal{R}}$ and $\tau_{\mathcal{M}} = \tau$ using $\mathcal{R}$ as follows. Given $(n, e) \leftarrow \mathcal{RSA}(1^k)$, $y \leftarrow \mathbb{Z}_n$ and a signing oracle limited to ℓ queries, $\mathcal{M}$ runs $\mathcal{R}$ on input (n, e, y) and simulates ℓ executions of $\mathcal{A}$ by making requests to the signing oracle. Whenever $\mathcal{R}$ calls $\mathcal{A}(n, e, m)$ for some $m \in \{0, 1\}^{\mathsf{m}}$, $\mathcal{M}$ forwards m to the signing oracle to get $s \in \mathcal{S}_{n,e}(m)$ and returns s to $\mathcal{R}$. Since $\mathcal{R}$ conforms to Game 0 and never calls $\mathcal{A}$ on (n', e', m) with $(n', e') \neq (n, e)$, $\mathcal{M}$ succeeds in simulating $\mathcal{A}$ perfectly. This means that for any τ, ℓ,

$$\mathsf{Succ}^0\left(\mathsf{UF\text{-}KOA}[\mathcal{S}], \tau, \ell\right) \leq \mathsf{Succ}\left(\ell\text{-}\mathsf{RE\text{-}CMA}[\mathcal{S}], \tau\right) .$$

Stage 2. We now use the definition of instance-non-malleability and Lemma 1. Obviously, there exists a trivial reduction $\mathcal{T}$ from $\mathsf{UF\text{-}KOA}[\mathcal{S}]$ to $\mathsf{INV}[\mathcal{RSA}]$: given arbitrary (n, e, m), $\mathcal{T}$ simply defines $y = \mu(m, r)$ for some randomly selected $r \leftarrow \{0, 1\}^{\mathsf{r}}$, requests $s = \mathsf{RSA}(n, e, y)$ and outputs s. $\mathcal{T}$ requires exactly one random selection and one evaluation of μ, which takes time $t_{\mathsf{UF\text{-}KOA}[\mathcal{S}]} =$

$\mathsf{poly}(k)$. Applying Lemma 1, one gets for any τ, ℓ and $\varepsilon \in (0,1)$:

$$\begin{aligned}\mathsf{Succ}\left(\mathsf{INV}\left[\mathcal{RSA}\right] \Leftarrow \mathsf{UF\text{-}KOA}\left[\mathcal{S}\right], \varepsilon, \tau, \ell\right) &\\ \leq \mathsf{Succ}^{0}\left(\mathsf{UF\text{-}KOA}\left[\mathcal{S}\right], \tau + \ell \cdot \mathsf{poly}(k), \ell\right) + \Delta^{\mathsf{INV}}_{\mathcal{RSA}}\left(\tau + \ell \cdot \mathsf{poly}(k), \ell\right) &\\ \leq \mathsf{Succ}\left(\ell\text{-}\mathsf{RE\text{-}CMA}\left[\mathcal{S}\right], \tau + \ell \cdot \mathsf{poly}(k)\right) + \Delta^{\mathsf{INV}}_{\mathcal{RSA}}\left(\tau + \ell \cdot \mathsf{poly}(k), \ell\right) .&\end{aligned}$$

We extend asymptotically the above to $\tau, \ell = \mathsf{poly}(k)$. Since by assumption $\mathsf{Succ}\left(\mathsf{INV}\left[\mathcal{RSA}\right] \Leftarrow \mathsf{UF\text{-}KOA}\left[\mathcal{S}\right], \varepsilon, \tau, \ell\right)$ is non-negligible and the instance-non-malleability of $\mathcal{RSA}$ imposes $\Delta^{\mathsf{INV}}_{\mathcal{RSA}}\left(\tau + \ell \cdot \mathsf{poly}(k), \ell\right)$ to remain negligible, one gets that $\mathsf{Succ}\left(\ell\text{-}\mathsf{RE\text{-}CMA}\left[\mathcal{S}\right], \tau + \ell \cdot \mathsf{poly}(k)\right)$ must be non-negligible. □

4.2 Existential Unforgeability and Known-Message Security

We now move on to the study of $\mathsf{EF\text{-}KOA}\left[\mathcal{S}\right]$ where again $\mathcal{S}$ is a real-life RSA signature scheme relying on an instance-non-malleable generator $\mathcal{RSA}$ and some padding function μ.

Theorem 2. *Let $\mathcal{S} = (\mathcal{RSA}, \mu)$ be an RSA-based signature scheme and assume that $\mathcal{RSA}$ is instance-non-malleable. If $\mathsf{EF\text{-}KOA}\left[\mathcal{S}\right] \equiv \mathsf{INV}\left[\mathcal{RSA}\right]$ then $\mathsf{RE\text{-}KMA}\left[\mathcal{S}\right]$ is polynomial.*

Proof. We build a meta-reduction $\mathcal{M}$ which converts an algorithm $\mathcal{R}$ playing Game 0 with respect to $\mathcal{A} \triangleq \mathsf{EF\text{-}KOA}\left[\mathcal{S}\right]$ into an algorithm which breaks $\mathsf{RE\text{-}KMA}\left[\mathcal{S}\right]$ and conclude using Lemma 1 as above. Assume $\mathcal{R}$ $(\varepsilon_{\mathcal{R}}, \tau)$-inverts $\mathcal{RSA}$ given access to an oracle $\mathcal{A}$ that breaks $\mathsf{EF\text{-}KOA}\left[\mathcal{S}\right]$ with probability one and assume that $\mathcal{R}$ runs $\mathcal{A}$ at most ℓ times. We build an algorithm $\mathcal{M}$ which $(\varepsilon_{\mathcal{M}}, \tau_{\mathcal{M}})$-breaks $\ell\text{-}\mathsf{RE\text{-}KMA}\left[\mathcal{S}\right]$ with $\varepsilon_{\mathcal{M}} \geq \varepsilon_{\mathcal{R}}$ and $\tau_{\mathcal{M}} = \tau$. Given $(n, e) \leftarrow \mathcal{RSA}(1^k)$, $y \leftarrow \mathbb{Z}_n$ and ℓ arbitrary message-signature pairs $(m_1, s_1), \ldots, (m_\ell, s_\ell)$, $\mathcal{M}$ runs $\mathcal{R}$ on (n, e, y) and simulates ℓ executions of $\mathcal{A}$ using the message-signature pairs: when $\mathcal{R}$ runs $\mathcal{A}(n, e)$ for $i = 1, \ldots, \ell$, $\mathcal{M}$ simply returns (m_j, s_j). $\mathcal{M}$ succeeds in simulating $\mathcal{A}$ perfectly, meaning that

$$\mathsf{Succ}^{0}\left(\mathsf{EF\text{-}KOA}\left[\mathcal{S}\right], \tau, \ell\right) \leq \mathsf{Succ}\left(\ell\text{-}\mathsf{RE\text{-}KMA}\left[\mathcal{S}\right], \tau\right) .$$

We then invoke the instance-non-malleability of $\mathcal{RSA}$ to conclude as in the second stage of the proof of Theorem 1. □

4.3 Impossible Proofs for RSA Signatures

Theorem 3. *Let $\mathcal{RSA}$ be an instance-non-malleable generator. There is no real-life RSA signature scheme $\mathcal{S} = (\mathcal{RSA}, \mu)$ such that $\mathsf{EF\text{-}CMA}\left[\mathcal{S}\right]$, $\mathsf{UF\text{-}CMA}\left[\mathcal{S}\right]$ or $\mathsf{EF\text{-}KMA}\left[\mathcal{S}\right]$ is equivalent to $\mathsf{INV}\left[\mathcal{RSA}\right]$ unless $\mathsf{INV}\left[\mathcal{RSA}\right]$ is polynomial.*

This is a direct corollary of Theorems 1 and 2 and the straightforward black-box reductions between security notions. A most striking consequence is that real-life RSA signatures that are proven to be existentially unforgeable under chosen

message attack under the RSA assumption in the RO model (*i.e.*, pretty much all of the ones currently used in cryptographic applications) do not verify that property in the standard model. This in turn tells us that these signature schemes separate the RO model from the standard model in a strong sense, unless their key generation escapes instance-non-malleability as per Definition 1.

5 On the Impossibility of Basing Unforgeability on the One-More RSA Assumption

A most crucial challenge that arises from Theorem 3 is to establish the real security level of common RSA signatures. If one cannot expect to reach a maximal security level $\mathsf{EF\text{-}CMA}[\mathcal{S}] \equiv \mathsf{INV}[\mathcal{RSA}]$, is there any hope to connect the unforgeability of these signatures to stronger assumptions? This section explores the feasibility of basing the unforgeability of $\mathcal{S}$ on the One-More RSA assumption. Here again, we report impossibilities. In particular, it is shown that PSS-R, FDH and PSS do not admit proofs of universal unforgeability under the One-More RSA assumption in the standard model if one assumes a slightly stronger notion of instance-non-malleability.

5.1 Extending Instance-Non-Malleability to $\mathsf{OM}[\mathcal{RSA}]$

We extend the previous definition of instance-non-malleability as follows. Instead of extracting roots, the algorithm $\mathcal{R}$ playing either Game 0 or Game 1 attempts to solve t-$\mathsf{OM}[\mathcal{RSA}]$ where $t \geq 0$ is an additional parameter. Again, the resources of $\mathcal{R}$ are modelled as a perfect oracle $\mathcal{A}(n, e, \cdot)$ where $\mathcal{A}$ is trivially reducible to $\mathsf{INV}[\mathcal{RSA}]$. The success probability of $\mathcal{R}$ is defined as

$$\mathsf{Succ}_t^i(\mathcal{R}, \mathcal{A}, \tau, \ell) = \Pr\left[\begin{array}{c}(n,e,d) \leftarrow \mathcal{RSA}(1^k) \\ y_0, \ldots, y_t \leftarrow \mathbb{Z}_n\end{array} : \begin{array}{c}\mathcal{R}^{\mathcal{O}_i}(n, e, y_0, \ldots, y_t) \\ = (y_0^d \bmod n, \ldots, y_t^d \bmod n)\end{array}\right],$$

where $i \in \{0,1\}$, $\mathcal{O}_0 = \{\mathcal{A}(n,e,\cdot)\}$, $\mathcal{O}_1 = \{\mathcal{A}(n,e,\cdot), \mathsf{RSA}(\cdot,\cdot,\cdot)\}$, $\mathcal{R}$ runs in time at most τ, totalizes at most ℓ queries to the oracles in $\mathcal{O}_i$ and requests made to $\mathsf{RSA}(\cdot,\cdot,\cdot)$ are of the form (n', e', y) with $(n', e') \in \mathsf{Range}\left(\mathcal{RSA}(1^k)\right)$, $(n', e') \neq (n, e)$ and $y \in \mathbb{Z}_{n'}$. We define

$$\Delta_{\mathcal{RSA}}^{t\text{-}\mathsf{OM}}(\tau, \ell) = \max_{\mathcal{A} \Leftarrow \mathsf{INV}[\mathcal{RSA}]} \left| \max_{\mathcal{R}} \mathsf{Succ}_t^1(\mathcal{R}, \mathcal{A}, \tau, \ell) - \max_{\mathcal{R}} \mathsf{Succ}_t^0(\mathcal{R}, \mathcal{A}, \tau, \ell) \right|$$

where the inner maxima are taken over all probabilistic algorithms $\mathcal{R}$ playing Game 1 and Game 0 respectively and the outer maximum is taken over all computational problems $\mathcal{A}$ trivially reducible to $\mathsf{INV}[\mathcal{RSA}]$ in polytime. We say that $\mathcal{RSA}$ is instance-non-malleable with respect to $\mathsf{OM}[\mathcal{RSA}]$ when $\Delta_{\mathcal{RSA}}^{t\text{-}\mathsf{OM}}(\tau, \ell) = \mathsf{negl}(k)$ when $t, \tau, \ell = \mathsf{poly}(k)$. Note that the previous definition is captured by setting $t = 0$. Lemma 1 still applies here under the following reformulation: for any positive integers t, τ, ℓ and any $\varepsilon \in (0,1)$,

$$\mathsf{Succ}\left(t\text{-}\mathsf{OM}[\mathcal{RSA}] \Leftarrow \mathcal{A}, \varepsilon, \tau, \ell\right) \leq \mathsf{Succ}_t^0\left(\mathcal{A}, \tau + \ell \cdot t_{\mathcal{A}}, \ell\right) + \Delta_{\mathcal{RSA}}^{t\text{-}\mathsf{OM}}\left(\tau + \ell \cdot t_{\mathcal{A}}, \ell\right) .$$

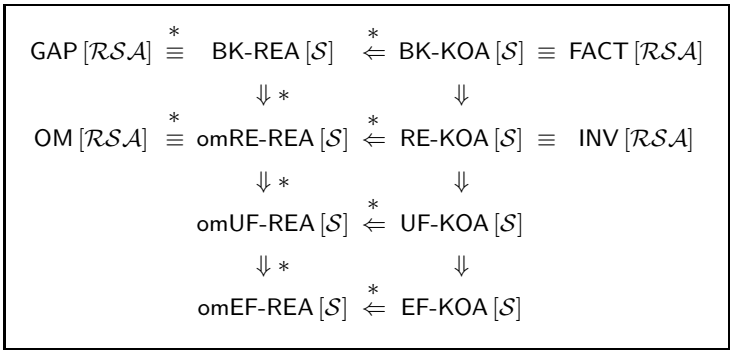

Fig. 2. Black-box reductions among security notions involving root extraction attacks

5.2 Introducing Root Extraction Attacks

We now introduce a stronger attack model wherein the adversary is given direct access to root extraction instead of a signing oracle. We define a notion of universal forgery under this type of attack, noting this security level $\mathsf{omUF\text{-}REA}\,[\mathcal{S}]$. A probabilistic algorithm breaks ℓ-$\mathsf{omUF\text{-}REA}\,[\mathcal{S}]$ if, given a random public key (n, e), $\ell+1$ random messages $m_0, \ldots, m_\ell$ and oracle access to $\mathsf{RSA}(n, e, \cdot)$, it can produce signatures $s_i \in \mathcal{S}_{n,e}(m_i)$ for $i = 0, \ldots, \ell$ in no more than ℓ requests to the oracle. More precisely, $\mathcal{A}$ is said to (ε, τ)-break ℓ-$\mathsf{omUF\text{-}REA}\,[\mathcal{S}]$ when

$$\Pr\left[\begin{array}{c}(n,e,d) \leftarrow \mathcal{RSA}(1^k) \\ m_0, m_1, \ldots, m_\ell \leftarrow \{0,1\}^{\mathsf{m}}\end{array} : \begin{array}{c}\mathcal{A}^{\mathsf{RSA}(n,e,\cdot)}(n, e, m_0, \ldots, m_\ell) = L \wedge \\ L \in \mathcal{S}_{n,e}(m_0) \times \cdots \times \mathcal{S}_{n,e}(m_\ell)\end{array}\right] \geq \varepsilon\ ,$$

where the probability is taken over the random coins of $\mathcal{A}$ and of the experiment, $\mathcal{A}$ runs in time τ and sends at most ℓ requests to $\mathsf{RSA}(n, e, \cdot)$. For completeness, we define ℓ-$\mathsf{omEF\text{-}REA}\,[\mathcal{S}]$ in a similar way, allowing the adversary to freely choose messages $m_0, \ldots, m_\ell$ with the restriction that no two message-signature pairs given as output are identical. We also introduce ℓ-$\mathsf{omRE\text{-}REA}\,[\mathcal{S}]$ where the adversary attempts to extract $\ell + 1$ e-th roots modulo n in no more than ℓ requests to $\mathsf{RSA}(n, e, \cdot)$. We see that for any real-life RSA signature scheme $\mathcal{S}$ and $\ell \geq 0$,

$$\ell\text{-}\mathsf{omEF\text{-}REA}\,[\mathcal{S}] \Leftarrow \ell\text{-}\mathsf{omUF\text{-}REA}\,[\mathcal{S}] \Leftarrow \ell\text{-}\mathsf{omRE\text{-}REA}\,[\mathcal{S}] \equiv \ell\text{-}\mathsf{OM}\,[\mathcal{RSA}]\ .$$

We also introduce ℓ-$\mathsf{BK\text{-}REA}\,[\mathcal{S}]$ where the adversary attempts to compute d from (n, e) by making at most ℓ calls to $\mathsf{RSA}(n, e, \cdot)$. We plot on Fig. 2 the relations holding between these security levels. Black-box reductions marked with a $*$, somewhat unfamiliar, are detailed in the full version of this paper [14].

5.3 Transposable and Derandomized Paddings

Transposable Paddings. Given a padding function μ mapping $\{0,1\}^{\mathsf{m}} \times \{0,1\}^{\mathsf{r}}$ to $\{0,1\}^{\mathsf{w}}$, we define μ^T over $\{0,1\}^{\mathsf{r}} \times \{0,1\}^{\mathsf{m}}$ by $\mu^T(m, r) = \mu(r, m)$. Thus μ^T

is obtained from μ by reversing the message and random domains of μ. We say that μ is *transposable* if μ^T is verifiable. When μ is transposable, μ^T is a proper padding function that defines a transposed signature scheme $\mathcal{S}^T$ allowing one to sign r-bit messages under m bits of randomness. To remain consistent with the definition of a signature scheme, we impose that $\mathsf{r} > 0$ for transposable paddings, resulting in that only probabilistic paddings are transposable. We say that $\mathcal{S}$ is transposable when its padding function is transposable. Obviously, $(\mathcal{S}^T)^T = \mathcal{S}$.

Derandomized Paddings. If μ is a (verifiable) padding function with domain and range as above, we define $\mu^\circ : \{0,1\}^{\mathsf{m}+\mathsf{r}} \to \{0,1\}^{\mathsf{w}}$ by setting $\mu^\circ(m) = \mu(m_0, m_1)$ where $|m_0| = \mathsf{m}$, $|m_1| = \mathsf{r}$ and $m = m_0 \| m_1$. Thus, μ° is a (verifiable) padding with no randomness and we call it the *derandomized* version of μ. The scheme $\mathcal{S}^\circ$ induced by μ° is said to be the derandomized version of $\mathcal{S}$. Note that $(\mathcal{S}^\circ)^\circ = \mathcal{S}^\circ$.

5.4 Impossibility Results for Transposable Paddings

Let us now consider the standard-model security level $\mathsf{UF\text{-}KOA}[\mathcal{S}]$ of a real-life scheme $\mathcal{S}$ combining an instance-non-malleable (*w.r.t* $\mathsf{OM}[\mathcal{RSA}]$) instance generator $\mathcal{RSA}$ with some transposable padding function μ.

Theorem 4. *If* $\mathsf{OM}[\mathcal{RSA}] \Leftarrow \mathsf{UF\text{-}KOA}[\mathcal{S}]$ *then* $\mathsf{omUF\text{-}REA}[\mathcal{S}^T]$ *is polynomial.*

Proof. We actually give a proof that if one had t-$\mathsf{OM}[\mathcal{RSA}] \Leftarrow \mathsf{UF\text{-}KOA}[\mathcal{S}]$ then $(t+\ell)$-$\mathsf{omUF\text{-}REA}[\mathcal{S}^T]$ could be broken for polynomial ℓ, for any t. This proof is not just a generalization of the one of Theorem 1 to the extended case $t \geq 0$. Let us assume that some reduction algorithm $\mathcal{R}$ converts a perfect universal forger $\mathcal{A}$ into an algorithm which $(\varepsilon_{\mathcal{R}}, \tau)$-breaks t-$\mathsf{OM}[\mathcal{RSA}]$ while playing Game 0 as per Section 5.1 in no more than ℓ calls to $\mathcal{A}$. Here again, we build a meta-reduction $\mathcal{M}$ converting $\mathcal{R}$ into an algorithm which $(\varepsilon_{\mathcal{M}}, \tau_{\mathcal{M}})$-breaks $(t+\ell)$-$\mathsf{omUF\text{-}REA}[\mathcal{S}^T]$ with $\varepsilon_{\mathcal{M}} \geq \varepsilon_{\mathcal{R}}$ and $\tau_{\mathcal{M}} = \tau_{\mathcal{R}} + \mathsf{poly}(t, \ell, k)$ and we conclude using the assumption that $\mathcal{RSA}$ is instance-non-malleable with respect to $\mathsf{OM}[\mathcal{RSA}]$.

Overview of $\mathcal{M}$. $\mathcal{M}$ attempts to break $(t+\ell)$-$\mathsf{omUF\text{-}REA}[\mathcal{S}^T]$. Given a random public key $(n, e) \leftarrow \mathcal{RSA}(1^k)$, a root extraction oracle $\mathsf{RSA}(n, e, \cdot)$ which $\mathcal{M}$ can call $t+\ell$ times and $t+\ell+1$ random messages $\overline{m}_0, \ldots, \overline{m}_{t+\ell} \leftarrow \{0,1\}^{\mathsf{r}}$, $\mathcal{M}$ must output $t+\ell+1$ signatures $\overline{s}_0, \ldots, \overline{s}_{t+\ell}$ on $\overline{m}_0, \ldots, \overline{m}_{t+\ell}$ with respect to $\mathcal{S}^T$. A description of $\mathcal{M}$ is as follows. $\mathcal{M}$ randomly selects $u_i \leftarrow \mathbb{Z}_n$ and sets $y_i = \mu(0, \overline{m}_i) \cdot u_i^e \bmod n$ for $i = 0, \ldots, t$. Then $\mathcal{M}$ runs $\mathcal{R}(n, e, y_0, \ldots, y_t)$. Since $\mathcal{R}$ complies with Game 0, $\mathcal{R}$ does not run $\mathcal{A}$ over $(n', e') \neq (n, e)$. $\mathcal{M}$ simulates ℓ executions of $\mathcal{A}$ by making requests to the root extraction oracle $\mathsf{RSA}(n, e, \cdot)$ to generate forgeries $s_1, \ldots, s_\ell$. These simulations will also result in the generation of signatures $\overline{s}_{t+1}, \ldots, \overline{s}_{t+\ell}$ on $\overline{m}_{t+1}, \ldots, \overline{m}_{t+\ell}$ with respect to $\mathcal{S}^T$. $\mathcal{R}$ may also send up to t queries to the root extraction oracle, which $\mathcal{M}$ perfectly simulates by forwarding the queries to its own root extractor. Overall, $\mathcal{M}$ makes at most $t+\ell$ queries. If $\mathcal{R}$ outputs $(y_0^d \bmod n, \ldots, y_t^d \bmod n)$, $\mathcal{M}$ sets $\overline{s}_i = y_i^d u_i^{-1} \bmod n$ for $i = 0, \ldots, t$ and returns $\overline{s}_0, \ldots, \overline{s}_{t+\ell}$, thereby succeeding in solving $(t+\ell)$-$\mathsf{omUF\text{-}REA}[\mathcal{S}^T]$.

Simulation of $\mathcal{A}$. Wlog, $\mathcal{R}$ runs $\mathcal{A}(n, e, m_i)$ for $i = 1, \ldots, \ell$. To simulate the i-th oracle call, $\mathcal{M}$ computes $\mu(m_i, \overline{m}_{t+i})$ and uses the root extractor $\mathsf{RSA}(n, e, \cdot)$ to get $\overline{s}_{t+i} = \mu(m_i, \overline{m}_{t+i})^d \bmod n$. $\mathcal{M}$ then sets $s_i = \overline{s}_{t+i}$. Note that $s_i = \overline{s}_{t+i}$ is simultaneously a valid signature on m_i with respect to $\mathcal{S}$ and a valid signature on $\overline{m}_{t+i}$ with respect to $\mathcal{S}^T$. $\mathcal{M}$ returns s_i to $\mathcal{R}$ as the output of $\mathcal{A}(n, e, m_i)$.

Summing up. The distributions of (n, e), $y_0, \ldots, y_t$ and the signatures comply with the definition of $\mathcal{R}$. $\mathcal{M}$ queries $\mathsf{RSA}(n, e, \cdot)$ no more than $t+\ell$ times. Because $\mathcal{M}$ simulates $\mathcal{A}$ perfectly, we have $\varepsilon_{\mathcal{M}} \geq \varepsilon_{\mathcal{R}}$. The extra time of $\mathcal{M}$ amounts to at most $t + \ell$ evaluations of μ and a few operations modulo n. This means that for any t, τ, ℓ,

$$\mathsf{Succ}_t^0\left(\mathsf{UF\text{-}KOA}\left[\mathcal{S}\right], \tau, \ell\right) \leq \mathsf{Succ}\left((t+\ell)\text{-}\mathsf{omUF\text{-}REA}\left[\mathcal{S}^T\right], \tau + \mathsf{poly}\left(t, \ell, k\right)\right) \ .$$

This gives the expected result using the above reformulation of Lemma 1 and assuming $\mathcal{RSA}$ is instance-non-malleable with respect to $\mathsf{OM}\left[\mathcal{RSA}\right]$. □

Most interestingly, it is the case that $\mathsf{omUF\text{-}REA}\left[\mathcal{S}^T\right]$ may remain unbroken in the RO model even though $\mathsf{UF\text{-}KOA}\left[\mathcal{S}\right]$ and $\mathsf{INV}\left[\mathcal{RSA}\right]$ are shown to be equivalent. A typical example of this separation is PSS-R, as we now see.

Definition 2. *Let* $H : \{0,1\}^{\mathsf{m}+\mathsf{r}} \to \{0,1\}^{\mathsf{t}}$ *and* $G : \{0,1\}^{\mathsf{t}} \to \{0,1\}^{\mathsf{w}-1-\mathsf{t}}$ *be two hash functions where* $\mathsf{w}, \mathsf{m}, \mathsf{r}, \mathsf{t}$ *are length parameters conforming to [17]. Define* $\text{PSS-R} = (\mathcal{RSA}, \mu)$ *where* $\mu(m, r) = 0 \parallel w \parallel (r \| m) \oplus G(w)$ *with* $w = H(m \| r)$.

Lemma 2 (RO-model REA security of PSS-R). PSS-R *is transposable and* $\mathsf{omUF\text{-}REA}\left[\left(\text{PSS-R}^T\right)^{G,H}\right] \equiv \mathsf{OM}\left[\mathcal{RSA}\right]$.

The proof of Lemma 2 is given in the full version of this work [14]. Theorem 4 shows that it is impossible to have $\mathsf{OM}\left[\mathcal{RSA}\right] \Leftarrow \mathsf{UF\text{-}KOA}\left[\mathcal{S}\right]$ for any transposable RSA signature scheme $\mathcal{S} = (\mathcal{RSA}, \mu)$ such that $\mathcal{RSA}$ is instance-non-malleable *w.r.t* $\mathsf{OM}\left[\mathcal{RSA}\right]$ and $\mathsf{omUF\text{-}REA}\left[\mathcal{S}^T\right]$ is intractable. This impossibility readily extends to all forms of unforgeability due to the natural black-box ordering between security notions. In particular, provided that $\mathsf{omUF\text{-}REA}\left[\text{PSS-R}^T\right]$ is intractable, which seems a reasonable assumption, there is no hope one can ever prove that PSS-R is unforgeable (in any sense) under the One-More RSA assumption.

5.5 Impossibility Results for Non-transposable Paddings

Theorem 4 ascertains that the unforgeability of $\mathcal{S}$ cannot be proven under the assumption that $\mathsf{OM}\left[\mathcal{RSA}\right]$ is intractable provided that $\mathcal{S}$ is transposable and assuming that $\mathsf{omUF\text{-}REA}\left[\mathcal{S}^T\right]$ is intractable. However this is not satisfactory since $\mathcal{S}$ may not be transposable. In this section, we alleviate the assumption that $\mathcal{S}$ is transposable. The price to pay is that we cannot provide evidence that $\mathsf{UF\text{-}KOA}\left[\mathcal{S}\right] \not\equiv \mathsf{OM}\left[\mathcal{RSA}\right]$ anymore. What we prove however is that $\mathsf{EF\text{-}KOA}\left[\mathcal{S}\right] \not\equiv \mathsf{OM}\left[\mathcal{RSA}\right]$ under the same assumptions.

Theorem 5. *If* $\mathsf{OM}[\mathcal{RSA}] \Leftarrow \mathsf{EF\text{-}KOA}[\mathcal{S}]$ *then* $\mathsf{omUF\text{-}REA}[\mathcal{S}^\circ]$ *is polynomial.*

Proof. We prove that if t-$\mathsf{OM}[\mathcal{RSA}] \Leftarrow \mathsf{EF\text{-}KOA}[\mathcal{S}]$ then $(t+\ell)$-$\mathsf{omUF\text{-}REA}[\mathcal{S}^\circ]$ can be broken for polynomial ℓ. Assume that some reduction algorithm $\mathcal{R}$ converts an existential forger $\mathcal{A}$ into an algorithm which $(\varepsilon_{\mathcal{R}}, \tau)$-breaks t-$\mathsf{OM}[\mathcal{RSA}]$ while playing Game 0 as per Section 5.1 in no more than ℓ calls to $\mathcal{A}$. We show how to build a meta-reduction $\mathcal{M}$ converting $\mathcal{R}$ into an algorithm which $(\varepsilon_{\mathcal{M}}, \tau_{\mathcal{M}})$-breaks $(t+\ell)$-$\mathsf{omUF\text{-}REA}[\mathcal{S}^\circ]$ with $\varepsilon_{\mathcal{M}} \geq \varepsilon_{\mathcal{R}}$ and $\tau_{\mathcal{M}} \approx \tau_{\mathcal{R}}$ and we conclude using the instance-non-malleability of $\mathcal{RSA}$ *w.r.t* $\mathsf{OM}[\mathcal{RSA}]$ as above.

Overview of $\mathcal{M}$. $\mathcal{M}$ attempts to break $(t+\ell)$-$\mathsf{omUF\text{-}REA}[\mathcal{S}^\circ]$. Given a random public key $(n,e) \leftarrow \mathcal{RSA}(1^k)$, a root extractor $\mathsf{RSA}(n,e,\cdot)$ which $\mathcal{M}$ can call $t+\ell$ times and $t+\ell+1$ random messages $\overline{m}_0, \ldots, \overline{m}_{t+\ell} \leftarrow \{0,1\}^{\mathsf{r}+\mathsf{m}}$, $\mathcal{M}$ must output the (unique) $t+\ell+1$ signatures $\overline{s}_0, \ldots, \overline{s}_{t+\ell}$ on $\overline{m}_0, \ldots, \overline{m}_{t+\ell}$ with respect to $\mathcal{S}^\circ$. A description of $\mathcal{M}$ is as follows. $\mathcal{M}$ randomly selects $u_i \leftarrow \mathbb{Z}_n$, sets $y_i = \mu(\overline{m}_i) \cdot u_i^e \bmod n$ for $i = 0, \ldots, t$ and runs $\mathcal{R}(n, e, y_0, \ldots, y_t)$. Since $\mathcal{R}$ conforms to Game 0, $\mathcal{R}$ never runs $\mathcal{A}$ over $(n', e') \neq (n, e)$. $\mathcal{M}$ simulates ℓ executions of $\mathcal{A}$ by making requests to the root extraction oracle $\mathsf{RSA}(n,e,\cdot)$ to generate arbitrary forgeries $(m_1, s_1), \ldots, (m_\ell, s_\ell)$. These simulations will also result in the generation of signatures $\overline{s}_{t+1}, \ldots, \overline{s}_{t+\ell}$ on $\overline{m}_{t+1}, \ldots, \overline{m}_{t+\ell}$ with respect to $\mathcal{S}^\circ$. $\mathcal{R}$ may also send up to t queries to its root extraction oracle, which $\mathcal{M}$ perfectly simulates by forwarding the queries to its own root extraction oracle $\mathsf{RSA}(n,e,\cdot)$. Overall, $\mathcal{M}$ makes at most $t+\ell$ queries. If $\mathcal{R}$ outputs $(y_0^d \bmod n, \ldots, y_t^d \bmod n)$, $\mathcal{M}$ sets $\overline{s}_i = y_i^d u_i^{-1} \bmod n$ for $i = 0, \ldots, t$ and returns $\overline{s}_0, \ldots, \overline{s}_{t+\ell}$, thereby succeeding in solving $(t+\ell)$-$\mathsf{omUF\text{-}REA}[\mathcal{S}^\circ]$.

Simulation of $\mathcal{A}$. Wlog, $\mathcal{R}$ runs $\mathcal{A}(n,e)$ for $i = 1, \ldots, \ell$. At the i-th call, $\mathcal{M}$ computes $\mu(\overline{m}_{t+i})$ and uses the root extractor to get $\overline{s}_{t+i} = \mu(\overline{m}_{t+i})^d \bmod n$. $\mathcal{M}$ then sets $s_i = \overline{s}_{t+i}$ and sets m_i to the m-bit prefix of $\overline{m}_{t+i}$ *i.e.,* $\overline{m}_{t+i} = m_i \| r_i$ for some r-bit string r_i. Note that $s_i = \overline{s}_{t+i}$ is a valid signature on $\overline{m}_{t+i}$ with respect to $\mathcal{S}^\circ$. $\mathcal{M}$ then returns (m_i, s_i) to $\mathcal{R}$ as the output of $\mathcal{A}(n,e)$.

Summing up. The distributions of (n,e), $y_0, \ldots, y_t$ and the signatures comply with the definition of $\mathcal{R}$. $\mathcal{M}$ queries $\mathsf{RSA}(n,e,\cdot)$ no more than $t+\ell$ times. The simulation of $\mathcal{A}$ being perfect, one gets $\varepsilon_{\mathcal{M}} \geq \varepsilon_{\mathcal{R}}$. The extra time of $\mathcal{M}$ amounts to at most $t+\ell$ evaluations of μ and a few operations modulo n. This implies that for any t, τ, ℓ,

$$\mathsf{Succ}_t^0\left(\mathsf{EF\text{-}KOA}[\mathcal{S}], \tau, \ell\right) \leq \mathsf{Succ}\left((t+\ell)\text{-}\mathsf{omUF\text{-}REA}[\mathcal{S}^\circ], \tau + \mathsf{poly}(t, \ell, k)\right) .$$

This gives the wanted result invoking again the reformulated Lemma 1 and assuming $\mathcal{RSA}$ is instance-non-malleable with respect to $\mathsf{OM}[\mathcal{RSA}]$, exactly as in the proof of Theorem 4. □

Application to FDH. Theorem 5 is perfectly suited to the case of FDH since $\mathrm{FDH}^\circ = \mathrm{FDH}$. The padding function μ is defined as $\mu(m,r) = H(m)$ where H is a w-bit hash function, $\mathsf{m} = *$ and $\mathsf{r} = 0$. We claim that

$$\mathsf{omUF\text{-}REA}\left[\mathrm{FDH}^{H}\right] \equiv \mathsf{OM}\left[\mathcal{RSA}\right] \ ,$$

as long as $n2^{-\mathsf{w}}$ remains polynomial. As $\mathsf{w} \simeq |n|$ in practice, $n2^{-\mathsf{w}}$ is bounded by a small constant. A proof of that fact is given in [14]. Under the assumption that $\mathsf{omUF\text{-}REA}\left[\mathrm{FDH}\right]$ is intractable in the standard model, we thus obtain that $\mathsf{EF\text{-}KOA}\left[\mathrm{FDH}\right]$ cannot be equivalent to $\mathsf{OM}\left[\mathcal{RSA}\right]$. This assumes of course that $\mathcal{RSA}$ is instance-non-malleable with respect to $\mathsf{OM}\left[\mathcal{RSA}\right]$.

Application to PSS. The signature scheme PSS exists in several versions [17,12]. We consider the version put forward by PKCS #1 v2.1.

Definition 3 (EMSA-PSS). *Let* $H : \{0,1\}^{\mathsf{m}} \rightarrow \{0,1\}^{\mathsf{t}_1}$, $F : \{0,1\}^{\mathsf{t}_2+\mathsf{t}_1+\mathsf{r}} \rightarrow \{0,1\}^{\mathsf{t}3}$ *and* $G : \{0,1\}^{\mathsf{t}_3} \rightarrow \{0,1\}^{\mathsf{w}-\mathsf{t}_3-9}$ *be three hash functions where* $\mathsf{w}, \mathsf{m}, \mathsf{r}, \mathsf{t}_1, \mathsf{t}_2$ *and* t_3 *are length parameters conforming to [17]. Define* $\mathrm{PSS} = (\mathcal{RSA}, \mu)$ *where* $\mu(m,r) = 0 \,\|\, \left(0^{\mathsf{w}-\mathsf{r}-\mathsf{t}_3-10} \,\|\, 1 \,\|\, r\right) \oplus G(w) \,\|\, w \,\|\, \mathtt{0xBC}$ *with* $w = F\left(0^{\mathsf{t}_2} \,\|\, H(m) \,\|\, r\right)$.

Lemma 3. $\mathsf{omUF\text{-}REA}\left[\left(\mathrm{PSS}^{\circ}\right)^{F,G,H}\right] \equiv \mathsf{OM}\left[\mathcal{RSA}\right]$.

The proof of Lemma 3 is given in the full paper [14]. Again, under the quite reasonable assumption that $\mathsf{omUF\text{-}REA}\left[\mathrm{PSS}^{\circ}\right]$ is intractable in the standard model, we have that $\mathsf{EF\text{-}KOA}\left[\mathrm{PSS}\right]$ cannot be equivalent to $\mathsf{OM}\left[\mathcal{RSA}\right]$ unless $\mathsf{OM}\left[\mathcal{RSA}\right]$ is easy (which in turn would contradict by itself the assumption that $\mathsf{omUF\text{-}REA}\left[\mathrm{PSS}^{\circ}\right]$ is intractable) or unless $\mathcal{RSA}$ is instance-malleable with respect to $\mathsf{OM}\left[\mathcal{RSA}\right]$.

Remark 1 (On basing unforgeability on $\mathsf{GAP}\left[\mathcal{RSA}\right]$*).* Finally, we note that under the very same assumptions, the unforgeability of RSA signatures cannot be based on $\mathsf{GAP}\left[\mathcal{RSA}\right]$ either. This is easily seen *reductio ad absurdum* as having $\mathsf{GAP}\left[\mathcal{RSA}\right] \Leftarrow \mathsf{UF\text{-}KOA}\left[\mathcal{S}\right]$ or $\mathsf{GAP}\left[\mathcal{RSA}\right] \Leftarrow \mathsf{EF\text{-}KOA}\left[\mathcal{S}\right]$ mechanically implies $\mathsf{OM}\left[\mathcal{RSA}\right] \Leftarrow \mathsf{UF\text{-}KOA}\left[\mathcal{S}\right]$ or $\mathsf{OM}\left[\mathcal{RSA}\right] \Leftarrow \mathsf{EF\text{-}KOA}\left[\mathcal{S}\right]$ by virtue of $\mathsf{OM}\left[\mathcal{RSA}\right] \Leftarrow \mathsf{GAP}\left[\mathcal{RSA}\right]$, which contradicts either Theorem 4 or Theorem 5.

6 Conclusion

This paper put forward several new impossibility results for RSA-based signature schemes. Among other results, we have shown that no real-life RSA signatures as per the definition of Section 2.3 that are based on instance-non-malleable key generation can be chosen-message secure under any one of the RSA, the gap RSA or the One-More RSA assumptions in the standard model. A challenging direction for future research would be to formally *confirm* or *refute* that common key generators are instance-non-malleable using computational number theory.

Acknowledgements. I would like to thank Jonathan Katz for his patience and suggestions that substantially improved the quality of this paper. This work has been financially supported by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.

References

1. M. Bellare, A. Boldyreva and A. Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In EUROCRYPT 2004, LNCS 3027, pp. 171–188, 2004.
2. M. Bellare, C. Namprempre, D. Pointcheval and M. Semanko. The One-More-RSA-Inversion Problems and the security of Chaum's Blind Signature Scheme. Journal of Cryptology, Vol. 16, No. 3, 2003, pp. 185-215.
3. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In ACM CCS 93, pp. 62–73, 1993. ACM Press.
4. A. Boldyreva and M. Fischlin. Analysis of random oracle instantiation scenarios for OAEP and other practical schemes. In CRYPTO 2005, LNCS 3493, pp. 412–429.
5. D. R. L. Brown. Unprovable security of RSA-OAEP in the standard model, 2006. Available at `http://eprint/iacr.org/2006/223`.
6. R. Canetti, O. Goldreich and S. Halevi. On the random-oracle methodology as applied to length-restricted signature schemes. In TCC 2004, LNCS 2951, pp. 40–57, 2004.
7. J-S. Coron. On the exact security of full domain hash. In CRYPTO 2000, LNCS 1880, pp. 229–235, 2000.
8. Y. Dodis, R. Oliveira and K. Pietrzak. On the generic insecurity of the full domain hash. In CRYPTO 2005, LNCS 3493, pp. 449–466, 2005.
9. C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. In CRYPTO'94, LNCS 839, pp. 234–246,1994.
10. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In CRYPTO'86, LNCS 263, pp. 186–194, 1987.
11. S. Goldwasser, S. Micali and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Comp., 17(2):281–308, 1988.
12. IEEE P1363a Committee. IEEE P1363a / D9 — Standard specifications for public key cryptography: Additional techniques. Document available at `http://grouper.ieee.org/groups/1363/index.html/`, 2001. Draft Version 9.
13. J. B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In CRYPTO 2002, LNCS 2442, pp. 111–126.
14. P. Paillier. Instance-non-malleable RSA-based cryptography, 2006. Available at `http://eprint/iacr.org/2006/.`
15. P. Paillier and D. Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In ASIACRYPT 2005, LNCS 3788, pp. 1–20, 2005.
16. P. Paillier and J. Villar. Trading one-wayness against chosen-ciphertext security in factoring-based encryption. In ASIACRYPT 06, LNCS 4284, pp. 252–266, 2006.
17. PKCS #1 v2.1: RSA cryptography standard (draft), document available at `http://www.rsasecurity.com/rsalabs/pkcs/`. RSA Data Security Inc., Sept. 2005.
18. M. O. Rabin. Digital signatures and public key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT, January 1979.
19. M. Tompa and H. Woll. Random self-reducibility and zero-knowledge interactive proofs of possession of information. UCSD TR CS92-244, 1992.
20. H. C. Williams. A modification of the RSA public-key encryption procedure. In IEEE Transactions on Information Theory, IT-26(6):726–729, 1980.
21. Adam Young and Moti Yung. *Malicious Cryptography: Exposing Cryptovirology.* John Wiley & Sons, first edition, 2005.

Selecting Secure Passwords

Eric R. Verheul

PricewaterhouseCoopers Advisory, Radboud University Nijmegen,
Institute for Computing and Information Sciences, P.O. Box 85096,
3508 AB Utrecht, The Netherlands
eric.verheul@[nl.pwc.com, cs.ru.nl]

Abstract. We mathematically explore a model for the shortness and security for passwords that are stored in hashed form. The model is implicitly in the NIST publication [8] and is based on conditions of the Shannon, Guessing and Min Entropy. We establish various new relations between these three notions of entropy, providing strong improvements on existing bounds such as the McEliece-Yu bound from [7] and the Min entropy lowerbound on Shannon entropy [3]. As an application we present an algorithm generating near optimally short passwords given certain security restrictions. Such passwords are specifically applicable in the context of one time passwords (e.g. initial passwords, activation codes).

1 Introduction

Consider the context of a computer system to which the logical access by a user (a human or another computer) is protected by a password. The threat that we consider is compromise of this password through one of the following two types of attack. In the *on-line guessing* type of attack, the attacker repeatedly makes guesses of the password, most likely first, and tests them by attempting to logon to the system. In our model the system has implemented "account lockout", locking the system after a certain number, say b, unsuccessful logon attempts which limits the effectiveness of the attack. In the *off-line guessing* type of attack, the attacker gets hold of some test-data from the system that enables him to test password guesses, most likely first, on his own systems. This information could for instance be a UNIX "passwd" file, a Windows SAM database or, more generally, the secure hash of a password. We distinguish two kinds of off-line attacks. In a *complete* attack the attacker is willing to take all the required computational effort to completely finish his attack algorithm thereby surely finding the password. In an *incomplete* attack the attacker is only willing to take a certain computational effort, a number of L guesses, in the attack, thereby finding the password only with a certain probability. To illustrate, suppose that an attacker has the SHA-1 hash of the password. If the attacker is willing to let the guess process run on a 1 GHz Pentium machine for a day this means that he is willing to perform about 2^{36} tries (cf. [2]); one might find it acceptable that the probability of success is at most 1%.

The central problem of this paper deals with choosing passwords that on the one hand have the functional requirement that they are "small" and on the

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 49–66, 2007.

other hand have the security requirement that they are "adequately" resistent against both on-line as off-line attacks, both complete as incomplete. Such passwords are specifically applicable in the context of one time passwords (e.g. initial passwords, activation codes).

Outline of the Paper
In Section 2 we describe the mathematical model we use for the central problem in this paper. In this model the Shannon entropy is taken as a measure for "smallness" of passwords, the Guessing entropy [6] as a measure for security of passwords against complete off-line attacks and the Min entropy (cf. [3]) as a measure for security of passwords against incomplete off-line attacks. In Section 3 we discuss and apply some techniques for calculating extreme points of convex sets. In Section 4 we present general, new relations between the three types of entropy. This provides both strong improvements on the McEliece-Yu bound from [7] and the Min entropy lowerbound [3] on Shannon entropy. In Section 5 we arrive at a new lower bound on the Shannon entropy of distributions, i.e. on the minimal length of the corresponding passwords, given restrictions on the Guessing and Min entropy. As an application we present in Section 6 an algorithm generating near optimally short passwords under these conditions that are specifically applicable in the context of one time passwords (e.g. initial passwords, activation codes). Finally Section 7 contains the conclusion of this paper and open problems.

Related Work
NIST Special Publication 800-63 [8] provides technical guidance in the implementation of electronic authentication, including passwords. The implicitly used mathematical model for security of passwords is similar to ours but the publication does not fully mathematically explore this model. Massey [6], cf. [5], shows that the entropy H for a password distribution is upper bounded in terms of its Guessing entropy α by $H \leq \log_2(e \cdot \alpha - 1)$. Note that Massey's bound is independent of the number of passwords n. By a counterexample it is indicated in [6] that no interesting lower bound on the entropy of a password exists in terms of the Guessing entropy alone. McEliece and Yu [7] show that $H \geq \frac{2\log_2(n)}{n-1}(\alpha - 1)$ indicating that such lower bounds exist if one takes into account the number of passwords n. It is well-known that the Shannon entropy is lower bounded by the Min entropy (cf. [3]), i.e., independent of n. Massey's bound can also be formulated as a lower bound on the Guessing entropy in terms of the Shannon entropy; in [1] Arikan provides another lower bound on the Guessing entropy in terms of the l^p-norm of the underlying probability distribution.

2 The Mathematical Model for Secure Passwords

In this section we describe our mathematical model for secure passwords selections. Further motivation of the model is placed in Appendix A. We assume that passwords correspond to a finite variable X with a discrete distribution $(p_1, \ldots, p_n)$ on n points (the number of possible passwords). That is, each $p_i \geq 0$

and they sum up to 1. To facilitate easy notation and formulae we assume throughout this paper that the probabilities p_i are denoted in a decreasing form, i.e., $p_1 \geq p_2 \ldots \geq p_n \geq 0$. The size of passwords is measured in our model by the *(Shannon) Entropy* $H(X)$ (or simply H) which is given by

$$H(X) = -\sum_{i=1}^{n} p_i \cdot \log_2(p_i),$$

and where we use the usual convention that $0 \cdot \log_2(0) = 0$. Our choice is motivated by the fact that the entropy corresponds with the average size of passwords in bits using an optimal coding for these passwords, most notably a coding based on a Huffman encoding tree [4]. The resistance against complete off-line attacks we measure by the *Guessing entropy*, cf. [6], denoted by $G(X)$ or simply α, given by

$$G(X) = \sum_{i=1}^{n} i \cdot p_i.$$

This relates to the expected number of tries for finding the password using an optimal strategy, i.e. trying the most likely keys first. Perhaps surprising, a large Guessing entropy by itself is not a sufficient guarantee for a secure passwords distribution. In [6] an example is given of a family of distributions (see also Section 4) that all have a fixed Guessing entropy but that have a highest probability that increases to one (and the entropy decreases to zero). That is, the probability that the first guess is successful goes to one implying that these distributions are certainly not "secure". From this example it is indicated that a viable security model also needs to take into account resistance against incomplete attacks. In our model, we measure this by the so-called *Min Entropy*, $H_\infty(X)$ or simply H_∞ given by $-\log_2(p_1)$, cf. [3]. If the Min entropy is sufficiently large, or equivalently p_1 sufficiently small, then one can assure that $\sum_{i=1}^{L} p_i \leq L \cdot p_1$ is sufficiently small too.

The resistance against on-line attacks is directly related to the probability of success in the optimal strategy, i.e. trying the b most likely keys. As typically the acceptable number b will be much smaller than the acceptable number L of an incomplete attack, we will only impose conditions on the effectiveness on the latter kind of attack. The central problem of this paper can now be mathematically formulated as follows: given lower bounds on the Guessing and Min entropy (or equivalently an upper bound on p_1) what is the minimal Shannon entropy possible and how can one calculate and apply minimal distributions?

3 Preliminaries

3.1 Extreme Points of Ordered Distributions with Fixed Guessing Entropy

We recall (cf. [9, p. 241]) that a point x of a convex set C is *extreme* if it is not an interior point of any line segment lying in K. Thus x is extreme iff

whenever $x = \lambda y + (1-\lambda)z$ with $0 < \lambda < 1$ we have $y \notin C$ or $z \notin C$. It is a direct consequence of the Krein-Milman theorem (cf. [9, p. 242]) that any closed, bounded convex set in $\mathbb{R}^n$ for some natural n is the convex hull of its extreme points.

Let r, s, n be natural numbers and let $f_1, ..., f_r$ and $F_1, ..., F_s$ be (linear) functionals on $\mathbb{R}^n$ and let $\delta_1, ..., \delta_r, \theta_1, ..., \theta_s \in \mathbb{R}$. The set $C \subset \mathbb{R}^n$ is defined by

$$C = \{x \in \mathbb{R}^n | f_i(x) = \delta_i \text{ for } i = 1, 2, ..., r \text{ and } F_j(x) \geq \theta_j \text{ for } j = 1, 2..., s\}.$$

Clearly, C is a closed convex set but is not necessarily bounded. Equations of type $f_i(x) = \delta_i$ we call *defining hyperplanes* and equations of type $F_j(x) \geq \theta_j$ we call *defining halfspaces*. We call a point x in C a *minimal intersection point* if

$$\{x\} = \cap_{i=1}^{r} f^{-1}(\delta_i) \bigcap \cap_{j \in S} F^{-1}(\theta_j) \tag{1}$$

for some subset S of $\{1, ..., s\}$. Determining the elements in (1) amounts to solving n variables based on $r + \|S\|$ equations as indicated in (1). Minimal intersection points then coincide with unique solutions x to such sets of equations that also satisfy the other conditions, i.e. lie in the remaining defining halfspaces. If each subset of n functionals in $\{f_1(\cdot), ..., f_r(\cdot), F_1(\cdot), ..., F_s(\cdot)\}$ is linearly independent (and so $r \leq n$ in particular), then one only needs to look at subset S of size $n - r$. The following result, using the notation of above, can be considered as part of the mathematical "folklore", cf. [11]. We provide proofs in Appendix B.

Theorem 1. *If C is bounded then the extreme points of C are precisely the minimal intersection points and C is the convex hull of the minimal intersection points.*

Let n be a natural number and α be a real number and let

$$C_{n,\alpha} = \{(p_1, ..., p_n) \in \mathbb{R}^n \mid \sum_{i=1}^{n} p_i = 1, \sum_{i=1}^{n} i p_i = \alpha, p_1 \geq p_2 ... \geq p_n \geq 0\}.$$

One can easily verify that $C_{n,\alpha} \neq \emptyset$ iff $1 \leq \alpha \leq (n+1)/2$ so from now on we implicitly assume that α satisfies the condition $1 \leq \alpha \leq (n+1)/2$. It is easily verified that $C_{n,1} = \{(1, 0, \ldots, 0)\}$ and $C_{n,(n+1)/2} = \{(1/n, 1/n, \ldots, 1/n)\}$.

Theorem 2. *The set $C_{n,\alpha}$ is a closed, bounded the convex set and is the hull of its extreme points. These extreme points take the form $X_{j,k,n}$ for integers j, k satisfying $1 \leq j \leq 2\alpha - 1 \leq k \leq n$ and*

$$\begin{array}{rcccccccccc} X_{j,k,n} = (& a_{j,k,n}, & a_{j,k,n}, & \cdots & a_{j,k,n}, & b_{j,k,n}, & \cdots & b_{j,k,n}, & 0, & \cdots & 0) \\ & & & & \uparrow & \uparrow & & \uparrow & \uparrow & & \\ & 1, & 2, & \cdots & j, & j+1, & \cdots & k, & k+1, & \cdots & n, \end{array}$$

where

$$a_{j,k,n} = \frac{-2\alpha + 1 + j + k}{j \cdot k}; b_{j,k,n} = \frac{2\alpha - (j+1)}{k(k-j)},$$

and where we define $b_{j,k,n} = 1/(2\alpha - 1)$ for $j = 2\alpha - 1 = k$ (which can only occur when $2\alpha - 1$ is an integer).

Proof: See Appendix B. □

Note that if in the previous theorem $2\alpha - 1$ is an integer then all points of type $X_{j,2\alpha-1,n}$ for $1 \leq j \leq 2\alpha-1$ are equal to the point whose first $2\alpha-1$ coordinates are equal to $1/(2\alpha-1)$ and the remaining ones are zero. We note that from the previous theorem it simply follows that the set $C_{n,\alpha}$ has more than one point if $\alpha < (n+1)/2$ and $n \geq 3$. So, for $n \geq 3$ it follows $C_{n,\alpha} = \{(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})\}$ iff $|C_{n,\alpha}| = 1$.

In practice often a certain number, say d, of the highest probabilities coincide. For instance when a dictionary is used and the d most likely passwords are chosen uniformly from a dictionary of size $d \leq n$. The set of such distributions takes the following form:

$$C_{n,\alpha,d} = \{(p_1, ..., p_n) \in \mathbb{R}^n \mid \sum_{i=1}^{n} p_i = 1, \sum_{i=1}^{n} i \cdot p_i = \alpha, \\ p_1 = p_2 \ldots = p_d \geq p_{d+1} \ldots \geq p_n \geq 0\}$$

We usually write $C_{n,\alpha}$ for $C_{n,\alpha,1}$. The following two results state some of the immediate properties of $C_{n,\alpha}$ and $C_{n,\alpha,d}$.

Proposition 1. *$C_{n,\alpha,d} \neq \emptyset$ iff $C_{n,\alpha} \neq \emptyset$ and $d \leq 2\alpha - 1$ iff $1 \leq \alpha \leq (n+1)/2$ and $d \leq 2\alpha - 1$.*

Proof: See Appendix B. □

Proposition 2. *We use the terminology and conditions of Theorem 2. The extreme points of $C_{n,\alpha}$ are the points $X_{j,k,n}$ satisfying $d \leq j \leq 2\alpha - 1 \leq k \leq n$.*

Proof: See Appendix B. □

3.2 Useful Formulae

We use the terminology of Theorem 2. It is convenient to introduce the function $G(x, y, z)$ with domain $\{(x, y, z) \in \mathbb{R}^3 \mid 0 < x \leq y \leq z\}$ given by

$$G(x, y, z) = -\left(\frac{(-y + x + z)}{z} \log_2(\frac{-y + x + z}{x \cdot z}) \right. \\ \left. + \frac{(y - x)}{z} \log_2(\frac{y - x}{z(z - x)})\right),$$

if $x < y \leq z$ and $G(y, y, z) = \log_2(y)$ and $G(y, y, y) = \log_2(y)$. Note that $G(x, z, z) = \log_2(z)$, $\lim_{x \uparrow y} G(x, y, z) = G(y, y, z)$ and that $G(j, 2\alpha - 1, k) = H(X_{j,k,n})$. Also note that values of $G(\cdot)$ are easily calculated. As usual we denote the entropy function on two points by $h(\cdot)$, i.e., $h(p) = -p \log_2(p) - (1-p) \log_2(1-p)$.

A real valued function $G(\cdot)$ from a convex set $C \subset \mathbb{R}^n$ is called *convex* (respectively *concave*) if $G(\lambda x + (1-\lambda)y) \leq \lambda G(x) + (1-\lambda)G(y)$ (respectively

$G(\lambda x + (1-\lambda)y) \geq \lambda G(x) + (1-\lambda)G(y)$ for all $x, y \in C$ and $0 \leq \lambda \leq 1$. Note that $G(\cdot)$ is convex iff $-G(\cdot)$ is concave. If $G(\cdot)$ is a twice-differentiable function in a single real variable (i.e., $C \subset \mathbb{R}$) then $G(\cdot)$ is convex (respectively concave) if $G^{''} \geq 0$ (respectively $G^{''} \leq 0$) on C. The proofs of the following two lemmas are straightforward verifications.

Lemma 1. *For $1 \leq x \leq y \leq z$ the following holds.*

1. $G(x,y,z) - \log_2(x) = G(1, y/x, z/x)$
2. $G(1,y,z) = h(\frac{y-1}{z}) + \frac{y-1}{z}\log_2(z-1)$

Lemma 2.

1. *For fixed x, z, the function $[x,z] \ni y \rightarrow G(x,y,z)$ is concave.*
2. *For fixed y, z, the function $[1,y] \ni x \rightarrow G(x,y,z)$ is concave.*
3. *For fixed x, y, the function $[y,\infty) \ni z \rightarrow G(x,y,z)$ increases to its maximum and is then decreasing.*

Proposition 3. $G(x,y,z) \geq \log_2(x) + \frac{y-x}{z-x}\log_2(z/x)$

Proof: By Lemma 2, for fixed x, z the function $[x,z] \ni y \rightarrow G(x,y,z)$ is concave. Note that for $y = x$ this function takes the value $\log_2(x)$ and for $y = x$ it takes the value $\log_2(z)$. If we write $y = (1-\lambda)x + \lambda z$ it follows that $\lambda = (y-x)/(z-x)$. From concavity it now follows

$$G(x,y,z) \geq (1-\lambda)\log_2(x) + \lambda\log_2(z) = \log_2(x) + \lambda\log_2(z/x),$$

from which the proposition follows. □

4 Relations Between Entropies

As explained in the introduction, we let passwords correspond to a finite variable X with a discrete distribution $(p_1, \ldots, p_n)$ on n points that we assume to be ordered, i.e., $p_1 \geq p_2 \ldots \geq p_n \geq 0$. The following inequalities hold, providing lower and upper bounds for the highest probability, i.e. p_1, of points in $C_{n,\alpha,d}$ in terms of α and n.

Theorem 3. *For $(p_1, \ldots, p_n) \in C_{n,\alpha,d}$ the following inequalities hold*

$$\frac{1}{2\alpha - 1} \leq \frac{-2\alpha + 1 + \lfloor 2\alpha - 1 \rfloor}{(\lfloor 2\alpha - 1 \rfloor)(\lceil 2\alpha - 1 \rceil)} + \frac{1}{\lfloor 2\alpha - 1 \rfloor} \leq p_1 \leq 1/d + 1/n - \frac{2\alpha - 1}{n}.$$

Proof: If $d = 1$ then from Theorem 2 it follows that the extreme points of $C_{n,\alpha,d}$ are of type $X_{j,k,n}$ with $1 \leq j \leq 2 \cdot \alpha - 1 \leq k \leq n$. It more generally follows from Proposition 2 that for $d > 1$, the extreme points of $C_{n,\alpha,d}$ are of type $X_{j,k,n}$ with $1 \leq j \leq 2 \cdot \alpha - 1 \leq k \leq n$.[1] It follows that $(p_1, \ldots, p_n)$ is in the convex hull of the extreme points $X_{j,k,n}$ with $d \leq j \leq 2 \cdot \alpha - 1 \leq k \leq n$.

[1] We distinguish between $d = 1$ and $d > 1$ to avoid a circular reasoning in the proofs of Propositions 1, 2 and Theorem 3.

Note that for fixed k the formula of the first coordinate of $X_{j,k,n}$, i.e., $a_{j,k,n}$, is decreasing in j. As the smallest permissible j equals d and the largest permissible equals $\lfloor 2\alpha - 1 \rfloor$, it follows that

$$\min\{a_{\lfloor 2\alpha-1\rfloor,k,n} \mid 2\alpha - 1 \leq k \leq n\} \leq p_1 \leq \max\{a_{d,k,n} \mid 2\alpha - 1 \leq k \leq n\} \quad (2)$$

Also note that for fixed j the formula of the first coordinate of $X_{j,k,n}$, i.e., $a_{j,k,n}$, is increasing in k. This means that the left hand side of (2) is equal to $a_{\lfloor 2\alpha-1\rfloor,\lceil 2\alpha-1\rceil,n}$, which is easily seen equal to

$$\frac{-2\alpha + 1 + \lfloor 2\alpha - 1 \rfloor}{(\lfloor 2\alpha - 1 \rfloor)(\lceil 2\alpha - 1 \rceil)} + \frac{1}{\lfloor 2\alpha - 1 \rfloor}.$$

That this expression is greater or equal to $1/(2\alpha - 1)$ follows from the easily verified inequality $x - \lfloor x \rfloor / (\lfloor x \rfloor \cdot \lceil x \rceil) + 1/\lfloor x \rfloor \geq 1/x$ for any $x \geq 0$. This concludes the proof for the second equality of the result. Similarly, the right hand side of (2) is equal to $a_{d,n,n}$ which is easily seen equal to $1/d + 1/n - \frac{2\alpha-1}{n}$ completing the proof of the theorem. □

As the function $-\log_2(\cdot)$ is decreasing, the bounds in the previous result can easily transformed in lower- and upperbounds for the Min entropy H_∞ in terms of α and n. The following result is a direct consequence; the right hand inequality also follows from a standard concavity result.

Corollary 1. $-\log_2(1 - \frac{2(\alpha-1)}{n}) \leq H_\infty \leq \log_2(2\alpha - 1)$

The next result enables the precise calculation of the minimum entropy on $C_{n,\alpha,d}$ that we denote by $M_{n,\alpha,d}$.

Theorem 4. *The following hold:*

1. $M_{n,\alpha,d} = \min\{G(j, 2\alpha - 1, k) \mid j \in \{d, \lfloor 2\alpha - 1 \rfloor\}, k \in \{\lceil 2\alpha - 1 \rceil, n\}\}$.
2. $M_{n,\alpha,d} \geq \min\{\log_2(2\alpha - 1), G(d, 2\alpha - 1, n)\} = \min\{\log_2(2\alpha - 1), h(\frac{2\alpha-1-d}{n}) + \frac{2\alpha-d}{n}\log_2(n/d - 1)\}$ *with equality if* $2\alpha - 1$ *is an integer.*

Proof: We prove the two parts of the theorem simultaneously. As the entropy function is concave, the minimum of the entropy function on $C_{n,\alpha,d}$ is the minimum the entropy achieves on its extreme points, i.e., the points of type $X_{j,k,n}$ with $d \leq 2\alpha - 1 \leq k \leq n$ (cf. Proposition 2), that is:

$$M_{n,\alpha,d} = \min\{G(j, 2\alpha - 1, k) \mid k, j \in \mathbb{N}, d \leq j \leq 2\alpha - 1 \leq k \leq n\}$$

From the concavity of the function $j \to G(j, 2\alpha - 1, k)$, i.e. Lemma 2, follows

$$M_{n,\alpha,d} = \min\{G(j, 2\alpha - 1, k) \mid j \in \{d, \lfloor 2\alpha - 1 \rfloor\}, k \in \mathbb{N}, 2\alpha - 1 \leq k \leq n\} \quad (3)$$

$$M_{n,\alpha} \geq \min\{G(j, 2\alpha - 1, k) \mid j \in \{d, 2\alpha - 1\}, k \in \mathbb{N}, 2\alpha - 1 \leq k \leq n\} \quad (4)$$

with equality if $2\alpha - 1$ is an integer. Finally, from equality (3) and the third part of of Lemma 2 we arrive at the first part of the theorem.

As $G(2\alpha-1,2\alpha-1,n) = G(d,2\alpha-1,2\alpha-1) = G(2\alpha-1,2\alpha-1,2\alpha-1) = \log_2(2\alpha-1)$ inequality (4) implies that

$$\begin{aligned} M_{n,\alpha,d} &\geq \min\{G(j,2\alpha-1,k) \mid j \in \{d,2\alpha-1\}, k \in \{2\alpha-1,n\}\} \\ &= \min\{\log_2(2\alpha-1), G(d,2\alpha-1,n)\} \end{aligned}$$

with equality if $2\alpha-1$ is an integer. The second part of the theorem now follows from combining the two formulae from Lemma 1. □

From Theorem 4 it follows that $M_{n,\alpha,d}$ is asymptotically equal to $G(d,2\alpha,n)$, i.e. to the entropy of the distribution the $X_{d,n,n}$ that goes to $\log_2(d)$. For $d=1$ this sequence actually forms the counterexample in [6] that no (interesting) lower bound on the entropy exists in terms of the Guessing entropy alone.

Theorem 4 also enables to determine that perhaps contrary to popular belief, the formula $\log_2(2\alpha-1) \leq H$ is actually only true for $n \leq 6$. Indeed, it is easily verified that the graph of the function $h_n : [1,(n+1)/2] \ni \alpha \to G(1,2\alpha-1,n)$ lies under the graph of $[1,(n+1)/2] \ni \alpha \to \log_2(2\alpha-1)$ for $n = 1,2,...,6$. From the second part of Theorem 4 it now follows that the formula is true for $n \leq 6$. That this formula does not hold for $n \geq 7$ also follows from the second part of Theorem 4 as $h_7(1.5) = 0.960953 < 1 = \log_2(2\alpha-1)$.

Theorem 5

$$M_{n,\alpha,d} \geq \log_2(d) + \frac{2\alpha-1-d}{n-d}\log_2(n/d)$$

This lowerbound on $M_{n,\alpha,d}$ is weaker than the lowerbound from the second part of Theorem 4. Moreover, both sides of the inequality are asymptotically equivalent in n.

Proof: Consider the following inequalities.

$$\begin{aligned} M_{n,\alpha,d} &\geq \min\left(G(d,2\alpha-1,n), \log_2(2\alpha-1)\right) \\ &\geq \min\left(\log_2(d) + \frac{2\alpha-1-d}{n-d}\log_2(n/d), \log_2(2\alpha-1)\right) \\ &\geq \log_2(d) + \frac{2\alpha-1-d}{n-d}\log_2(n/d), \end{aligned}$$

The first inequality is the second part of Theorem 4 and the second inequality follows from Proposition 3. For the last inequality; note that the function $[2\alpha-1,\infty) \ni n \to \log_2(d) + \frac{2\alpha-1-d}{n-d}\log_2(n/d)$, is decreasing. As this function converges to the value $\log_2(2\alpha-1)$ for $n \downarrow 2\alpha-1$, the last inequality follows and thereby the first two parts of the theorem.

With respect to the last part of the theorem; the sequence of distributions $X_{d,n,n}$ all have Guessing entropy equal to α and it is easily seen that their Shannon entropies converge to $\log_2(d)$. □

The previous result with $d=1$ is an extension of the McEliece-Yu bound from [7]. Its proof also shows that the lowerbound in the second part of Theorem 4

for $d = 1$ provides a stronger bound on $M_{n,\alpha,1}$ than the McEliece-Yu bound. It is easily shown that the difference between these bounds is about $h(2(\alpha-1)/n)$, about one in practice.

5 Secure Password Distributions

Let $(p_1, ..., p_n) \in C_{n,\alpha}$ and let $0 < \delta \leq 1$. Then $C_{n,\alpha,\delta}$ is the set:

$$\{(p_1, ..., p_n) \in \mathbb{R}^n \mid \sum_{i=1}^{n} p_i = 1, \sum_{i=1}^{n} i \cdot p_i = \alpha, \delta \geq p_1 \geq p_2 ... \geq p_n \geq 0\}.$$

From the proof of Theorem 3 it follows that the extreme point $X_{\lfloor 2\alpha-1 \rfloor, \lceil 2\alpha-1 \rceil, n} \in C_{n,\alpha}$ has a minimal first coordinate, namely $a_{\lfloor 2\alpha-1 \rfloor, \lceil 2\alpha-1 \rceil, n}$. So $C_{n,\alpha,\delta} \neq \emptyset$ iff $C_{n,\alpha} \neq \emptyset$ and $a_{\lfloor 2\alpha-1 \rfloor, \lceil 2\alpha-1 \rceil, n} \leq \delta$. Or in other words that

$$\frac{-2\alpha + 1 + \lfloor 2\alpha - 1 \rfloor}{(\lfloor 2\alpha - 1 \rfloor)(\lceil 2\alpha - 1 \rceil)} + \frac{1}{\lfloor 2\alpha - 1 \rfloor} \leq \delta \text{ and } \alpha \leq (n+1)/2. \tag{5}$$

Clearly, the fact that $C_{n,\alpha,\delta}$ is non-empty does not imply that it contains an element with first coordinate equal to δ. We call δ *admissible* if there is such an element. It simply follows from the proof of Theorem 3 that δ is admissible iff

$$\frac{-2\alpha + 1 + \lfloor 2\alpha - 1 \rfloor}{(\lfloor 2\alpha - 1 \rfloor)(\lceil 2\alpha - 1 \rceil)} + \frac{1}{\lfloor 2\alpha - 1 \rfloor} \leq \delta \leq 1 - \frac{2(\alpha-1)}{n}. \tag{6}$$

The following theorem discusses the extreme points of $C_{n,\alpha,\delta}$ and how to calculate them. For $v \in \mathbb{R}^m$ we let $(v)_1$ denote the first coordinate of v.

Theorem 6. *The set of points* E'

$$\{\lambda X_{j_1,k_1,n} + (1-\lambda) X_{j_2,k_2,n} \mid j_1 = j_2 \text{ or } k_1 = k_2, \lambda \in [0,1] : \\ \lambda (X_{j_1,k_1,n})_1 + (1-\lambda)(X_{j_2,k_2,n})_1 = \delta\} \cup \{f \in E \mid (f)_1 \leq \delta\}$$

is finite and its convex hull spans $C_{n,\alpha,\delta}$. *In particular, all extreme points of* $C_{n,\alpha,\delta}$ *are in* E'.

Proof: See Appendix B. □

Let $H(n, \alpha, \delta)$ denote $\min\{H(c) \mid c \in C_{n,\alpha,\delta}\}$. The following immediate result shows how $H(n, \alpha, \delta)$ can be precisely calculated.

Theorem 7. $H(n, \alpha, \delta)$ *is equal to the minimum value that the entropy takes on the set* E' *defined in Theorem 6, which is the minimum of at most* $\lfloor 2\alpha - 1 \rfloor * \lfloor n - 2\alpha - 1 \rfloor$ *real numbers.*

We extend the meaning of

$$a_{j,k,n} = \frac{-2\alpha + 1 + j + k}{j \cdot k}$$

from Theorem 2 to include any real $0 < j \leq 2\alpha - 1 \leq k \leq n$. The function $(0, 2\alpha - 1] \ni j \rightarrow a_{j,k,n}$ is decreasing and takes as an image the segment $[1/(2\alpha-1), \infty)$. The inverse of this function is a function $g_k^\alpha : [1/(2\alpha-1), \infty) \rightarrow (0, 2\alpha - 1]$ given by $g_k^\alpha(x) = \frac{k-2\alpha+1}{kx-1}$.

The following is the main result of this section. The idea of calculating an lower bound on the Shannon entropy given a Guessing entropy and α upperbound δ on the highest probability occurring is simple: just find the real number j such that the "virtual" extreme point $X_{j,n,n}$ has a first probability $a_{j,n,n}$ equal to δ. The lowerbound on the Shannon entropy is then the minimum of the entropy of the "virtual" extreme point and $\log_2(2\alpha - 1)$. The proof of Theorem 8 is skipped due to space restrictions but will be part of the full version of this paper.

Theorem 8. *Let α be fixed, $1 \leq 2\alpha - 1 \leq n$, and let δ be such that $C_{n,\alpha,\delta} \neq \emptyset$ (so in particular $\delta \geq 1/(2\alpha - 1)$, then*

$$H(n, \alpha, \delta) \geq \min(G(g_n^\alpha(\delta), 2\alpha - 1, n), \log_2(2\alpha - 1)).$$

Moreover, the sequence $\{\min(G(g_n^\alpha(\delta), 2\alpha - 1, n), \log_2(2\alpha - 1))\}_n$ is decreasing and converges to $-\log_2(\delta)$.

The following result is a consequence of Theorem 8 in an analogous fashion as Theorem 5 is a consequence of Theorem 4.

Theorem 9. *Let α be fixed, $1 \leq 2\alpha - 1 \leq n$, and let δ be such that $C_{n,\alpha,\delta} \neq \emptyset$ (so in particular $\delta \geq 1/(2\alpha - 1)$, then*

$$H(n, \alpha, \delta) \geq \log_2(g_n^\alpha(\delta)) + \frac{2\alpha - 1 - g_n^\alpha(\delta)}{n - g_n^\alpha(\delta)} \log_2(n/g_n^\alpha(\delta)). \tag{7}$$

This lowerbound on $H_{n,\alpha,\delta}$ is weaker than the lowerbound from the second part of Theorem 8. Moreover, the sequence $\{\log_2(g_n^\alpha(\delta)) + \frac{2\alpha-1-g_n^\alpha(\delta)}{n-g_n^\alpha(\delta)} \log_2(n/g_n^\alpha(\delta))\}_n$ is decreasing and converges to $-\log_2(\delta)$.

Corollary 2. *Let $(p_1, p_2, ..., p_n)$ be an (ordered) password distribution with Shannon entropy H, Guessing entropy α and Min entropy H_∞ then*

$$\begin{aligned} H &\geq \min(G(g_n^\alpha(p_1), 2\alpha - 1, n), \log_2(2\alpha - 1)) \\ &\geq \log_2(g_n^\alpha(p_1)) + \frac{2\alpha - 1 - g_n^\alpha(p_1)}{n - g_n^\alpha(p_1)} \log_2(n/g_n^\alpha(p_1)) \geq H_\infty. \end{aligned}$$

We make some remarks on Theorem 8. If we fill in the largest possible admissible δ in Theorem 8, i.e., $\delta = 1 - 2(\alpha - 1)/n$, we obtain the second part of Theorem 4 for $d = 1$ which itself an improvement of the McEliece-Yu bound by Theorem 5. The bound in Theorem 8 is strong in the sense that for δ that equal the first coordinate of an extreme point of type $X_{j,n,n}$, i.e. $\delta = a_{j,n,n}$ equality in Theorem 8 holds provided $H(X_{j,n,n}) \leq \log_2(2\alpha - 1)$. In Appendix C it is further indicated that the bound in Theorem 8 is strong and that taking the minimum with $\log_2(2\alpha - 1)$ cannot be relaxed. It is also indicated that the distributions $X_{j,n,n}$ are in fact "local" minima.

6 Selecting Near Optimal Secure Passwords

The strongness discussed above of Theorem 8 gives rise to the following algorithm, providing a near optimal password distribution with minimal Shannon entropy in our security model. In this algorithm we assume that the Guessing entropy α is an integer and that the bound on the highest occurring probability δ is of the form $1/D$ for some natural number D. These are very mild restrictions. For the existence of such distributions (cf. Theorem 3) one requires that $1/(2\alpha - 1) \leq \delta$. If the latter equality holds, the only distribution satisfying is the uniform one on $2\alpha - 1$ points and the (minimal) Shannon entropy equals the Guessing entropy. So we assume that $1/(2\alpha - 1) < \delta$, i.e. $D < 2\alpha - 1$. As $\lim_{n\to\infty} g_n(1/D) \uparrow D$ it also follows that

$$\lim_{n\to\infty} G(g_n(1/D), 2\alpha - 1, n) = \lim_{n\to\infty} G(D, 2\alpha - 1, n) = \log_2(D).$$

It now follows from Theorem 8 that when n grows to infinity, the minimum Shannon entropy $H(n, \alpha, \delta)$ decreases to $\log_2(D)$. For two reasons the distribution $X_{D,n,n}$ is an obvious choice for a finite approximation of this minimum. Firstly, its Shannon entropy converges to this minimum. Secondly, the highest probability occurring in $X_{D,n,n}$, i.e., $a_{D,k,n}$, is less than δ but can be taken arbitrarily close it. Moreover, as discussed at the end of Section 5, for n large enough the distribution $X_{D,n,n}$ establishes the minimum Shannon entropy in its own "class", i.e., the distributions on n points with Guessing entropy equal to α and highest probability $\leq a_{D,n,n}$.

The distributions of type $X_{D,n,n}$ also have a simple form: the first D coordinates are equal to $a_{D,n,n}$ and the remaining $n - D$ coordinates are equal to $b_{D,n,n}$. This makes generation of passwords in accordance with this distribution quite convenient by using a Huffman tree as follows. First generate a '0' with probability $P_{min} = D * a_{D,n,n}$ and a '1' with probability $P_{max} = (n-D) * b_{D,n,n}$. If 0 is generated, generate a random string of size D bits and concatenate it with '0', if '1' is generated then generate a random string of size $(n-D)$ bits and concatenate it with '1'. One can easily verify that the average size of such generated strings is at most the Shannon entropy of $X_{D,n,n}$ plus one bit. By increasing n one obtains password generation methods with average bit length arbitrarily close to $\log_2(D)$ whereby with a small probability (i.e., $(n - D) * b_{D,n,n}$ decreasing to zero) large passwords will occur of size $\log_2(n - D) \approx \log_2(n)$. In the table below we have placed some of the characteristic figures for $\alpha = 2^{64}$ and $\delta = 2^{-40}$.

If one applies this password generation method in a context where the system generates user passwords to be used repeatedly by the user, the user will be inclined to have changed the issued large password until the system proposes a small password. This of course undermines the security assumptions of the system. Also when using passwords repeatedly, it is important that they are easily memorable which the generated passwords in their current form are not. Consequently the password generation method described is only practically applicable when the passwords generated are One Time Passwords (OTPs). OTPs

arise in many applications such as in activation codes for digital services (e.g. prepaid mobile credit typically stored on a scratch card). Also initial computer passwords supplied by the IT department of an organization can be considered to be OTPs.

$\log_2(n)$	$-\log_2(a_{D,n,n})$	Average pwd length	Min length	Max length	P_{min}	P_{max}
65.0	65.00	65.00	40.0	65.0	2.98E-08	1.00E+00
65.5	41.77	58.90	40.0	65.5	2.92E-01	7.07E-01
66.0	41.00	54.00	40.0	66.0	5.00E-01	5.00E-01
66.5	40.62	50.30	40.0	66.5	6.46E-01	3.53E-01
67.0	40.41	47.56	40.0	67.0	7.50E-01	2.50E-01
67.5	40.28	45.53	40.0	67.5	8.23E-01	1.76E-01
68.0	40.19	44.04	40.0	68.0	8.75E-01	1.25E-01
68.5	40.13	42.95	40.0	68.5	9.11E-01	8.83E-02
69.0	40.09	42.14	40.0	69.0	9.37E-01	6.25E-02
69.5	40.06	41.56	40.0	69.5	9.55E-01	4.41E-02
70.0	40.04	41.13	40.0	70.0	9.68E-01	3.12E-02

7 Conclusion

We have presented a mathematical model for secure passwords and we have presented an algorithm providing near-optimal distributions in this model as well as a simple algorithm generating binary passwords accordingly. Such algorithms are specifically applicable in the context of one time passwords (e.g. initial passwords, activation codes). In addition we have established various new relations between the three notions of entropy (Shannon, Guessing, Min), providing strong improvements on existing bounds. Our results indicate that the expression $\log_2(2\alpha - 1)$, which we propose to call the *Searching* entropy, relates better to the other two entropies than the Guessing entropy α in its natural form.

It follows from Theorem 8 that distributions with fixed Guessing entropy α that satisfy $\log_2(2\alpha - 1) \leq H$ (an apparent popular belief) is of non-zero Lebesgue measure, i.e. the probability that a random distribution on n points with Guessing entropy equal to α satisfies this inequality is non-zero. It seems an interesting problem to establish the behavior of this probability in terms of α and n. A similar question is: what is the probability that a random distribution on n points satisfies $\log_2(2\alpha - 1) \leq H$? Based on our experiments it seems that this probability is close to one which we have actually shown for $n \leq 6$ as then all distributions satisfy this inequality.

Acknowledgments

Lisa Bloomer and Franklin Mendivil are thanked for discussing and eventually providing (Franklin) me with a technique for choosing random probability

distributions on n points used in my simulations. Frans van Buul is thanked for writing the initial simulation software, Christian Cachin for discussions on the various types of entropy, Berry Schoenmakers for providing me some initial mathematical techniques and Marcel van de Vel for the discussions on convexity.

References

1. E. Arikan, *An inequality on guessing and its application to sequential decoding*, IEEE Trans. Inform. Theory, vol. 42, pp. 99-105, 1996.
2. A. Bosselaers, *Even faster hashing on the Pentium*, rump session presentation at Eurocrypt97, May 13, 1997.
3. C. Cachin, *Entropy Measures and Unconditional Security in Cryptography*, volume 1 of ETH Series in Information Security and Cryptography. Hartung-Gorre Verlag, Konstanz, Germany, 1997 (Reprint of Ph.D. dissertation No. 12187, ETH Zrich).
4. D.A. Huffman, *A method for the construction of minimum-redundancy codes*, Proceedings of the I.R.E., 1952, pp. 1098-1102
5. D. Malone, W.G. Sullivan, *Guesswork and entropy*, IEEE Transactions on Information Theory, Volume 50, Issue 3, pp. 525-526, 2004.
6. J.L. Massey, *Guessing and entropy*, Proc. 1994 IEEE International Symposium on Information Theory, 1994, p.204.
7. R.J. McEliece, Z. Yu, *An inequality on entropy*, Proc. 1995 IEEE International Symposium on Information Theory, 1995, p.329.
8. NIST, *Electronic Authentication Guideline*, Special Publication 800-63, 2004.
9. H.L. Royden, *Real analysis*, Macmillan Publishing company, New York, 1988.
10. Sci.crypt crypto FAQ, `http://www.faqs.org/faqs/cryptography-faq/part04`.
11. M. L. J. van de Vel, *Theory of Convex Structures*, North-Holland, 1993.

A Appendix: Notes on the Model

Our model describes an ideal situation in which the computer system owner knows or can prescribe the probability distribution according to which users choose passwords. This is the case when the computer system owner generates the passwords for its users. In common practice, the system owner can at least highly influence the probability distribution by imposing "complexity rules" for passwords chosen by users.

In our model we have not taken the Shannon entropy as a measure for security of passwords which seems to be widely done. We have only found non-valid motivations for this, that are typically based on the misconception, e.g. in [8], [10], that the Shannon entropy and Guessing entropy are related by $\log_2(\alpha) = H$.

Perhaps contrary to popular belief, even an inequality of type

$$\log_2(2\alpha - 1) \leq H. \tag{8}$$

between the Shannon entropy and the Guessing entropy is not generally true as is shown by the earlier mentioned example of Massey [6].

In Theorem 8 we prove a variant on inequality (8) that does hold. We note that in this and many other results in this paper the expression $\log_2(2\alpha - 1)$ takes a prominent place. In fact, one can argue that this expression is a more suitable definition of Guessing entropy.

As Massey's bound can be rewritten as $\alpha \geq \log_2(H) - \log_2(e) \approx \log_2(H) - 1.4$ one can use the Shannon entropy in an underestimate for the Guessing entropy This indicates that the Shannon entropy can be used as an underestimate for resistance against complete off-line attacks. Without referring to Massey's bound, appendix A of [8] uses $\alpha \geq \log_2(H)$ and consequently from a theoretical perspective the numbers Table A.1 in [8] are about one bit too small. A large Shannon (and consequently Guessing) entropy does not provide resistance against incomplete attacks. To this end, for $0 < \delta < 1$ consider the distribution $(\delta, q_1, q_2, \ldots q_m)$ with $q_i = (1-\delta)/m$. This distribution has a Shannon entropy that goes to infinity when m goes to infinity, while the probability that the first guess is successful is δ irrespective of m. In other words, the Shannon and Guessing entropies alone are not an appropriate measure for secure password distribution as is suggested in table A.1 of [8].

In our model we have only considered an attacker that is after one specific password. In practice the attacker might test-data for several, say P, passwords (e.g. of different users), e.g. a UNIX "passwd" file or a Windows "SAM" database. If the attacker is only after one password (and not necessarily all of them or a specific one), his optimal strategy would be in parallel: trying if any of the passwords is the most likely password etcetera. If the test-data is just a secure hash of a password, then the attacker would be able to speed up the complete attack by about a factor P, which is why it is common practice to *salt* passwords before applying a secure hash function to it. If we assume that passwords are indeed adequately salted, the attacker does not gain any advantage from a parallel attack when he is aiming to mount a complete attack, i.e. finding all passwords. However the attacker gains advantage when he is aiming to mount an incomplete attack. Indeed if the attacker divides the computational effort he is willing to spend over the number of passwords, his probability of success is higher than if he only uses this effort to guess only one password. Our model can be used to quantify resistance against this attack as well by the following observation: if the probability of success of a incomplete attack with effort L against one password is q, then the probability of success of an incomplete P-parallel attack with effort $L \cdot P$ is $1 - (1-q)^P \approx q \cdot P$.

B Appendix: Proofs on Convexity

Proof of Theorem 1: Note that C is a closed set. It suffices to prove the first part of the result as the remainder then follows from the Krein-Milman theorem. To this end, let $x \in C$ be an extreme point. Let be $S \subset \{1, ..., s\}$ of highest cardinality such that

$$x \in \cap_{i=1}^{r} f^{-1}(\delta_i) \bigcap \cap_{j \in S} F^{-1}(\theta_j)$$

is of lowest affine dimension. Now suppose that this dimension is not zero, i.e. that x is not a minimal intersection point. Then first of all, $S \neq \{1, 2, ..., s\}$ as then set $\cap_{i=1}^{r} f^{-1}(\delta_i) \bigcap \cap_{j \in \{1,2,...,s\}} F^{-1}(\theta_j)$ would be an unbounded subset of C that is bounded by assumption. So $\{1, 2, ..., s\} \setminus S \neq \emptyset$ and for every $j \in \{1, 2, ..., s\} \setminus S$ it follows that $F_j(x) > \delta_j$. That is, there exists a small Euclidean ball B around x such that $F_j(y) > \delta_j$ for all $y \in B$. It now follows that

$$B \bigcap \cap_{i=1}^{r} f^{-1}(\delta_i) \bigcap \cap_{j \in \{1,2,...,s\}} F^{-1}(\theta_j) \subset C \tag{9}$$

Now, the intersection of a Euclidean ball and a affine space of dimension ≥ 1 contains more points than only its centre (i.e., x). Suppose that z is also in (9) than it easily follows that $x - (z - x) = 2x - z$ is also in (9). It then follows that $y = \frac{1}{2}(2y - z) + \frac{1}{2}z$ and x can not be an extreme point. We arrive at a contradiction and we conclude that x is a minimal intersection point.

Conversely, suppose that x is a minimal intersection point, i.e. there exists a S of $\{1, ..., s\}$, such that:

$$\{x\} = \cap_{i=1}^{r} f^{-1}(\delta_i) \bigcap \cap_{j \in S} F^{-1}(\theta_j). \tag{10}$$

Now suppose that x is not an extreme point, that is $x = \lambda y + (1 - \lambda)z$ with $0 < \lambda < 1$ and $y, z \in C \setminus \{x\}$. It simply follows that for each $j \in S$ we have $F_j(y) = F_j(z) = \theta_j$ as otherwise $F_j(x) > \theta_j$. We conclude that y, z are also elements of the left hand side of (10) contradicting that x is minimal intersection point. □

Proof of Theorem 2: The cases $n = 1$ and $n = 2$ can be easily verified directly and we assume that $n \geq 3$. Note that for $f_1(x) = \sum_{i=1}^{n} x_i$, $f_2(x) = \sum_{i=1}^{n} i \cdot x_i$, $F_j(x) = x_j - x_{j+1}$ for $1 \leq j < n$ and $F_n(x) = x_n$, the set $C_{n,\alpha}$ takes the form:

$$C_{n,\alpha} = \{x \in \mathbb{R}^n | f_1(x) = 1, f_2(x) = \alpha, \text{and } F_j(x) \geq 0 \text{ for } j = 1, 2..., n\}.$$

As $C_{n,\alpha}$ is clearly bounded we aim to use Theorem 1. For this we need to look for unique solutions of x of the equation $f_1(x) = 1, f_2(x) = \alpha$ and any subsets of the equations

$$F_1(x) = 0; F_2(x) = 0; ... F_n(x) = 0.$$

As any $n - 3$ or smaller subset of these equations will certainly not result in unique solutions x, we only need to consider any $n - 2$, $n - 1$ and n-subset of these equations.

An $(n-2)$-subset pertains to leaving out two equations, say the j-th and the k-th with $1 \leq j < k \leq n$ which leads to the following system of equations:

$$\begin{aligned} x_1 = x_2 = ... = x_j \\ x_{j+1} = x_{j+2} = ... = x_k \\ x_{k+1} = x_{k+2} = ... = x_n = 0. \end{aligned}$$

Together with the conditions $f_1(x) = 1, f_2(x) = \alpha$ simple calculations shows that this results in a unique solution $X_{n,j,k}$ as defined in the theorem. However these solutions also need to satisfy the remaining two equations, i.e., $x_j \geq x_{j+1}$ and $x_k \geq x_{k+1} = 0$. Simple calculations show that the first condition is equivalent to $k \geq 2\alpha - 1$ and that the second condition is equivalent to $j \leq 2\alpha - 1$. We conclude that the $X_{j,k,n}$ described in the theorem are all extreme points of $C_{n,\alpha}$.

An $(n-1)$-subset pertains to a $1 \leq j \leq n$ and

$$x_1 = x_2 = ... = x_j$$
$$x_{j+1} = x_{j+2} = ... = x_n = 0.$$

If this has a solution, then $2\alpha - 1$ must be equal to j and hence an integer. The first j coordinates of this solution will be equal to $1/(2\alpha - 1)$ and the remaining ones are zero. We arrive at $X_{2\alpha-1,2\alpha-1,n}$. Finally, an n-subset cannot results in solutions, let alone unique ones. □

Proof of Proposition 1: As the last equivalence is evident, we only prove the first one. To this end, let $(p_1, ..., p_n) \in C_{n,\alpha,d} \neq \emptyset$, then from Theorem 3 with $d = 1$ it follows that $1/(2\alpha - 1) \leq p_1$. If $(p_1, ..., p_n) \in C_{n,\alpha,d} \subset C_{n,\alpha}$ then clearly $p_1 \leq 1/d$. Hence it follows that $d \leq 2\alpha - 1$. The "only if" part of the first equivalence now follows from $C_{n,\alpha,d} \subset_{n,\alpha}$. Conversely suppose that $C_{n,\alpha} \neq \emptyset$ and $d \leq 2\alpha - 1$. Then according to Theorem 2 $X_{d,n,n}$ is one of the extreme points of Theorem 2 $C_{n,\alpha}$. It evidently follows that $X_{d,n,n} \in C_{n,\alpha,d}$, showing that this set is not empty. Actually, this alternatively follows from $X_{\lfloor 2\alpha-1 \rfloor,n,n} \in C_{n,\alpha,d}$. □

Proof of Proposition 2: We start the proof with two observations. First, from the description of the points $X_{j,k,n}$ in Theorem 2 it follows that all first j probabilities are strictly larger than the remaining ones provided that $k \neq 2\alpha - 1$. Second, if $k = 2\alpha - 1$ (hence $2\alpha - 1$ is an integer in particular), then for all $1 \leq j \leq 2\alpha - 1$, the points $X_{j,k,n}$ are equal to the distribution consisting of $2\alpha - 1$ non-zero probabilities equal to $1/(2\alpha - 1)$. As $d \leq 2\alpha - 1$ by Proposition 1 it evidently follows that then all first d probabilities are equal in $X_{j,k,n}$.

For a proof of the proposition; by using the description of the points $X_{j,k,n}$ in Theorem 2 it directly follows that if $d \leq j \leq 2\alpha - 1 \leq k$ that then the first d probabilities are equal.

Conversely, let a point $Y \in C_{n,\alpha}$ have its first d probabilities equal. Then by Theorem 2 the point Y is the convex combination of points $X_{j,k,n}$ satisfying $1 \leq j \leq 2\alpha - 1 \leq k$, $j < k$. Suppose that in this convex combination, some points $X_{j,k,n}$ contribute that do not satisfy $d \leq j$, i.e. $d > j$. If for some of these points $k = 2\alpha - 1$ then, by the second observation at the beginning of the proof, the contribution of this $X_{j,k,n}$ can be replaced with $X_{d,k,n}$.

So we may assume for the point $X_{j,k,n}$ contributing to Y satisfies $d > j$ and $k > 2\alpha - 1$. Let j' be the smallest j with this condition, it follows from the first observation at the beginning of the proof, that the first $j' < d$ probabilities in Y are strictly larger than the $j' + 1$-th probability and hence that the first d probabilities in Y are not equal. We arrive at a contradiction. □

Proof of Theorem 6: We number the points in E, i.e. $E = \{e_1, e_2, \ldots, e_{|E|}\}$. Clearly,

$$C_{n,\alpha,\delta} = \{\sum_{i=1}^{|E|} \lambda_i e_i \mid e_i \in E, \sum_{i=1}^{|E|} \lambda_i = 1, \lambda_i \geq 0, \sum_{i=1}^{|E|} \lambda_i (e_i)_1 \leq \delta\}. \tag{11}$$

We relate $C_{n,\alpha,\delta}$ with

$$L = \{(\lambda_1, \lambda_2, \ldots, \lambda_{|E|}) \mid \lambda_i \geq 0, \sum_{i=1}^{|E|} \lambda_i = 1, \sum_{i=1}^{|E|} \lambda_i (e_i)_1 \leq \delta\} \subset \mathbb{R}^{|E|}.$$

For $l = (\lambda_1, \lambda_2, \ldots, \lambda_{|E|}) \in L$ we define $l \cdot E = \sum_{i=1}^{|E|} \lambda_i \cdot e_i$. As the set L is convex, closed and bounded it is spanned by it extreme points. Following Theorem 1 the extreme points of L are of type

$$\begin{aligned} F = \{(\lambda_1, \lambda_2, \ldots, \lambda_{|E|}) \mid\ & \text{only two different } \lambda_i, \lambda_j \in [0,1] \text{ are non-zero and} \\ & \lambda_i + \lambda_j = 1, \lambda_i \cdot (e_i)_1 + \lambda_j \cdot (e_j)_1 = \delta \\ & \text{or } \lambda_i = 1 \text{ and } (e_i)_1 \leq \delta\} \end{aligned}$$

Clearly, F is finite. Also, any convex combination $(\lambda_1, \lambda_2, \ldots, \lambda_{|E|})$ occurring in (11) is a convex combination of elements in F. In other words, the convex hull of $F \cdot E$ is equal to $C_{n,\alpha,\delta}$. That is, an extreme point of $C_{n,\alpha,\delta}$ is either an extreme point f in $C_{n,\alpha}$ with $(f)_1 \leq \delta$ or of type

$$\lambda X_{j_1,k_1,n} + (1-\lambda) X_{j_2,k_2,n} \tag{12}$$

with $1 \leq j_1, j_2 \leq 2\alpha - 1 \leq k_1, k_2 \leq n$ and $\lambda \in (0,1)$.

We now take another view at the extreme points of $C_{n,\alpha,\delta}$. Using the technique used in the proof of Theorem 2 it follows that the extreme points of $C_{n,\alpha,\delta}$ are either $f \in E$ with $(f)_1 \leq \delta$ or take the form

$$\begin{array}{cccccccccccc} (\ \delta, & \delta, & \cdots \delta, & a, & \cdots a, & b, & \cdots & b, & 0, & \cdots 0) \\ & & \uparrow & \uparrow & \uparrow & \uparrow & & \uparrow & \uparrow & & \\ & & j, & j+1, & \cdots k, & k+1, & \cdots & m, & m+1 & \cdots\ n & \end{array} \tag{13}$$

with the condition that $\delta \geq a \geq b > 0$ and that this point is in $C_{n,\alpha}$. [2]

If either k_1 or k_2, say k_1, in expression (12) is equal to $2\alpha - 1$ (also implying that $2\alpha - 1$ is an integer) then this expression also holds if we take $j_1 = j_2$. Indeed, all points of type $X_{j,2\alpha-1,n}$ are equal (cf. Theorem 2). So assume that $2\alpha - 1 < k_1, k_2$. As is shown by Theorem 2 all extreme points $X_{j,k,n}$ with $2\alpha - 1 < k$ in E are of a special form: the first j coordinates are equal and strictly larger than the $j+1$ to k-th coordinates which are also equal and strictly larger than zero. It follows immediately that a point as in expression (12) can only be of the prescribed form (13) if either $j_1 = j_2$ or $k_1 = k_2$. □

[2] We note that, unlike in the proof of Theorem 2, not all points of the prescribed form (13), automatically satisfy the remaining conditions.

C Appendix: Comparison of Bounds

In Figure 1 below we have for $n = 23$ and $\alpha = 7$ depicted the graphs of $\delta \to H(n, \alpha, \delta)$ calculated using Theorem 7 labeled by "A"; $\delta \to \min(G(g_n^\alpha(\delta), 2\alpha - 1, n), \log_2(2\alpha - 1))$ labeled by "B", $\delta \to G(g_n^\alpha(\delta), 2\alpha - 1, n)$ labeled by "C", the bound in Theorem 9 labeled by "D" and the Min entropy $\delta \to -\log_2(\delta)$ labeled by "E". Finally we have depicted the 13 points $(a_{j,n,n}, H(X_{j,n,n}))$. It is easily verified that Theorem 8 is strong in the sense that for all δ that equal the first coordinate of an extreme point of type $X_{j,n,n}$, i.e. $\delta = a_{j,n,n}$ equality in Theorem 8 holds provided $H(X_{j,n,n}) \leq \log_2(2\alpha - 1)$. However, the figure below indicates that these distributions are actually "local" minima with respect to the Shannon entropy. The figure also indicates that the bound in Theorem 8 is strong, certainly in comparison with the bound in Theorem 9 and the Min entropy. We finally note that the example also shows that taking the minimum with $\log_2(2\alpha - 1)$ in Theorem 8 cannot be relaxed.

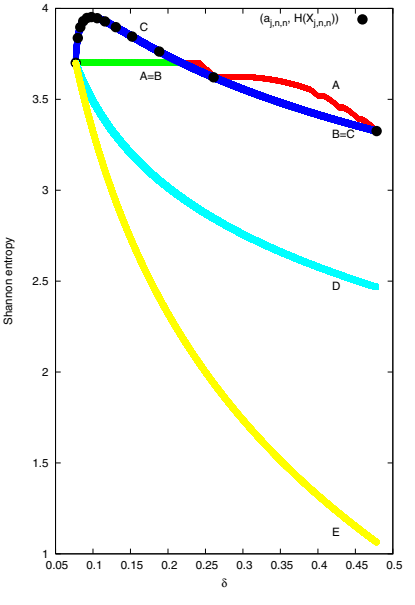

Fig. 1. Comparison of bounds

Human Identification Through Image Evaluation Using Secret Predicates*

Hassan Jameel[1], Riaz Ahmed Shaikh[1], Heejo Lee[2], and Sungyoung Lee[1]

[1] Department of Computer Engineering, Kyung Hee University, 449-701 Suwon, South Korea
{hassan, riaz, sylee}@oslab.khu.ac.kr,
[2] Department of Computer Science and Engineering, Korea University Anam-dong, Seongbuk-gu, Seoul 136-701, South Korea
heejo@korea.ac.kr

Abstract. The task of developing protocols for humans to securely authenticate themselves to a remote server has been an interesting topic in cryptography as a replacement for the traditional, less secure, password based systems. The protocols proposed in literature are based on some underlying difficult mathematical problem, which are tuned so as to make them easily computable by humans. As a result these protocols are easily broken when desired to be efficiently executable. We present a Human Identification Protocol based on the ability of humans to efficiently process an image given a secret predicate. It is a challenge-response protocol in which a subset of images presented satisfies a secret predicate shared by the challenger and the user. We conjecture that it is hard to guess this secret predicate for adversaries, both humans and programs. It can be efficiently executed by humans with the knowledge of the secret which in turn is easily memorable and replaceable. We prove the security of the protocol separately for human adversaries and programs based on two separate assumptions and justify these assumptions with the help of an example implementation.

1 Introduction

The problem of constructing human identification protocols is an important one in the cryptographic community. That is to say, how can a human H authenticate to a remote server C, without using any computational aid? To make matters worse, the communication link between H and C is controlled by an adversary who can either passively listen to their conversation or actively interfere at will. Under such conditions, it is desirable that this adversary should not be able to masquerade as H even after observing a number of authentication sessions.

* This research was supported by the MIC (Ministry of Informations and Communications), Korea under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment) incollaboration with SunMoon University. The corresponding author is Dr. Sungyoung Lee.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 67–84, 2007.

Notice that traditional password based schemes are completely insecure in this environment, in the sense that even the remote terminal which is being used by H to perform the authentication protocol is not trusted. It might contain malicious software like key-loggers, that can grab and record everything that H types, or someone might be using a hidden camera to see the alpha-numerics being typed. Ideally, H should be presented with a set of challenges, which H processes efficiently with the help of some shared secret and replies such that the responses yield little or nothing about the secret. The amount of processing required on H's part defines the feasibility of the protocol. What kind of protocol is practical for humans? We are not particularly good at numerical calculations; however we can be reasonably proud of our image processing abilities. If we can pose a challenge that involves evaluating an image based on some secret criteria then we might be able to construct a human executable protocol.

Consider the example of an IQ test based on images. The purpose of such a test is to check a person's intellectual abilities. The hardness of these questions is relative; for some these tests are harder than for others. But once a particular hint for a question is given, it would be answered promptly unlike someone who has to solve the question without any aid. If we assume that the hint to the question is a secret shared between the questioner and the human, then those who know the secret hint before hand can reply instantly whereas others without the knowledge of the hint would take considerably longer time. Furthermore, since we are involving images, the task of writing a computer program to answer these questions seems extremely hard as well, much like the automated CAPTCHA tests [6] that most humans can solve but computer programs cannot pass with a certain probability. The absence of such a "hint" makes the task of guessing the answer hard even for a human adversary. Additionally and more importantly, this "hint" can be renewed after even a small number of authentications as it will not be hard for a human to remember a natural statement.

The idea of constructing secure and efficiently executable human identification protocols using human's cognitive abilities about picture processing is not new. Matsumoto and Imai [1] were the first to propose a human identification protocol, followed by Wang *et al* [2], Matsumoto [3], Yang Li and Teng [4], Hopper and Blum [5] and Li [7]. All of these schemes can be implemented graphically. The idea is to map the secret with a set of images. The user only has to remember the images instead of a string of secret. This allows for easy secret recollection as humans can easily recall images as compared to textual strings. However, the security of the scheme relies on some underlying numerical problem like the Learning Parity with Noise (LPN) problem in [5] and the images are only used as an aid for learning the secret. Due to this fact, the complexity of these schemes is still a bit high, e.g. the HB protocol of [5] requires more than 300 seconds on average by humans. More recently, Weinshall [12] has proposed a scheme based on the SAT problem, which was subsequently broken in [13]. Notice that even in their scheme, the secret is a set of prespecified grid positions and in order to easily memorize these positions, pictures are introduced. They do not make use of the underlying structure of the images apart from using them as a memory aid. DeJa Vu[8] and Passface [9] are

purely graphical authentication schemes in which the user is asked to remember a subset of secret pictures and is then asked to authenticate by choosing his/her chosen images in a pool of pictures. A similar concept is to point and click secret locations in an image as in [10]. Apparently, in these schemes the user does not have to do any computational effort whatsoever. But this does not come without a drawback. These systems are not secure if someone is observing the actions of the user. If an adversary monitors the users' selections, it will be pretty easy to learn the secret images or locations. We argue that an image itself has a very complex structure and features, and instead of using it just as a memory aid, we can use the internal properties of images to pose challenges that can be efficiently executed by humans. Additionally, we might relax the required number of authentication sessions with a given secret, if the secret is easy to remember by a human and can be changed without too much effort on H's part. From the previous efforts, we can see that apart from the usual trade off between security, secret size and computational time, we face the problem that if we want to renew the secret after a small number of authentication sessions, it seems hard for a human to remember the new secret immediately.

In this paper, we show that it is possible to construct a challenge-response protocol, each challenge of which contains a kind of AI hard problem like CAPTCHA [6] for a computer algorithm. Moreover, the challenge is presented in such a way that a human without the knowledge of the secret can make "little sense" of the correlation between the elements of the challenge. We claim that such hard problems exist, and a protocol based on these problems will be very hard to break for adversaries, both humans and computer programs, while being efficiently executable for the legitimate user. Exact quantification of the hardness of these problems however remains an open issue.

2 An Informal Description of the Protocol

The central idea proposed in this paper is informally as follows: How can we combine the notion of CAPTCHA (creating a challenge-response that is not susceptible to bots) and secure user authentication that is not vulnerable to shoulder surfing or sniffing? To accomplish this feat, we propose the following protocol:

SETUP. User and the server agree upon a secret that is a composite of the following:

1. A simple question Q (which we will call a predicate) with only a binary answer, such as "Does the picture contain a woman?"
2. A set of distinct random numbers $a_1, a_2, \ldots, a_r$; all between 1 and n.

SERVER TO USER. A list of n pictures that are uniformly distributed with respect to the question Q above.

USER TO SERVER. n-bit binary string such that for pictures numbered $a_1, a_2, \ldots, a_r$ the corresponding bits are answers to the secret question Q and for all the other positions the bits are random.

The server accepts if the answer string is correct at the designated places. We can do the online step repeatedly to amplify security. In the full version of the protocol in Section 4, we permute the a_i's to make it harder for the adversary to extract any information from the answer string. The probability of guessing the correct permutation would be far less than that of guessing a correct random ordering of numbers. Security is based on the fact that (i) a bot or a computer program does not know the relationship between the pictures, and (ii) a human watching the proceedings would not know which bits are significant, which in turn will make it hard to guess the question being answered.

3 Definitions

We start with a set of definitions formalizing the notions of Identification Protocols and the new concepts introduced in this paper. We first define the notion of an *AI problem solver* which is modified from the definition of an *AI problem* in [6].

Definition 1. *An AI problem solver is a function $f : S \rightarrow R$, where S is a set of AI problem instances and $R \in \{0,1\}^*$ is the answer alphabet. A family of AI problem solvers is a map $F : \mathsf{Keys}(F) \times S \rightarrow R$. Here $\mathsf{Keys}(F)$ denotes the set of all AI problem solvers from S to R. Namely, if $k \in \mathsf{Keys}(F)$ then $F_k : S \rightarrow R$ is an AI problem solver.*

Notice that we specifically define a family of AI problem solvers instead of just a single one. Such a family will allow us to distribute different secrets, namely $k \in \mathsf{Keys}(F)$, to different users for authentication. The concept is similar to a function family.

We define the (δ, τ)-hardness of an AI problem solver similar to [6]:

Definition 2. *An AI problem solver f is said to be (δ, τ)-solved if there exists a program A, running in time at most τ, on an input $s \overset{R}{\leftarrow} S$, such that*

$$\Pr\left[s \overset{R}{\leftarrow} S : A(s) = f(s)\right] \geq \delta$$

f is said to be (δ, τ)-hard if no current program is a (δ, τ)solution to f, and the AI community agrees that it is hard to find such a solution.

Definition 3. *A family of AI problem solvers is said to be (δ, τ)-hard if for all keys $k \in \mathsf{Keys}(F)$, F_k is (δ_k, τ_k)-hard with $\delta_k \leq \delta$ and $\tau_k \leq \tau$.*

Definition 4. *We say that a function family, $F : \mathsf{Keys}(F) \times S \rightarrow R$ is $(\lambda(r), \tau)$-resilient against key recovery, if for all H running in time at most τ, we have:*

$$\begin{aligned}\Pr[&k \overset{R}{\leftarrow} \mathsf{Keys}(F); b_1 b_2 \ldots b_r \overset{R}{\leftarrow} \{0,1\}^r; \\ &s_1 s_2 \ldots s_r \leftarrow S | \, F_k(s_1) F_k(s_2) \ldots F_k(s_r) = b_1 b_2 \ldots b_r; \\ &k' \leftarrow H(s_1 s_2 \ldots s_r) : k = k'] < \lambda(r)\end{aligned}$$

Notice that H is not shown the value of the function F_k at each of the 'r' inputs. H only knows that the answer to each input belongs to the range R. It has to guess the correlation between the different inputs. Of course, the inputs must have an internal structure in order for the above definition to make sense. We do not elaborate this correlation between the inputs here as it will become clear when we describe the security of our protocol against human adversaries, instantiated with a suitable choice of the function family F in Section 6.

We restate the definitions of identification protocols and human executable protocols from [5] for reference:

Definition 5. *An Identification Protocol is a pair of probabilistic interactive programs (H, C) with shared auxiliary input z, such that the following conditions hold:*

- *For all auxiliary inputs z,* $\Pr\left[\langle H\left(z\right), C\left(z\right)\rangle = accept\right] > 0.9$
- *For each pair $x \neq y$,* $\Pr\left[\langle H\left(x\right), C\left(y\right)\rangle = accept\right] < 0.1$

When $\langle H, C\rangle = accept$, we say that H authenticates to C. The transcript of C contains challenges c and that of H comprises responses r to these challenges.

Definition 6. *An Identification Protocol (H, C) is (α, β, t)-human executable if at least a $(1-\alpha)$ portion of the human population can perform the calculations H unaided and without errors in at most t seconds with probability greater than $(1-\beta)$.*

Discussion on the definitions. We have separately defined a family of AI problem solvers and an $(\lambda\left(r\right), \tau)$-resilient function family. This partition is due to the fact that we want different security assumptions for a program and a human adversary. For an adversarial program, we require that the actual hardness in breaking our protocol relates to solving a function from the family of AI problem solvers. The set of keys '$\mathsf{Keys}(F)$' need not be hidden from this program or more strongly, the program might even be given the particular key being used and asked to guess the value of the function at a new input. We conjecture that such a program will still not be able to succeed but with a small probability. More details will follow in the next sections. On the other hand, we require a rather weak security assumption for human adversaries. Obviously, for a human the AI problem solvers will not pose any problems by definition. Instead, we present a $(\lambda\left(r\right), \tau)$-resilient function family to the human adversary. More specifically, we hide the set $\mathsf{Keys}(F)$ from the human adversary; randomly select a key from this set; draw a set of r inputs at random and ask the human adversary to guess the key. Since we assume the function family is $(\lambda\left(r\right), \tau)$-resilient, the probability of guessing the key is very small. We will present a function family F and argue that it satisfies both these requirements. The same function family F can both be a family of AI problem solvers and $(\lambda\left(r\right), \tau)$-resilient at the same time. Indeed, we give an example of such an F.

4 Proposed Protocol

The main theme of our protocol as described in Section 2 is to present a human with a series of pictures that satisfy a certain predicate. The user answers a pre-specified set of these pictures in an order determined by a hidden secret permutation. The assumption is that the analysis of these pictures by a computer program is extremely hard and even for a human adversary, the challenge of guessing the secret predicate when the answers are given in a random order, seems implausible. We first give a description of the protocol based on a generic building block F, and give a detailed practical example of this building block in Section 6.

PRELIMINARIES. The following notations and functions will be used in the protocol and hence forth.

$\mathsf{Perm}\,(L, l)$: Outputs the hidden permutation string $P = q_0^* p_1 q_1^* p_2 q_2^* \cdots p_l q_l^*$ of length L, obtained by first randomly selecting a permutation of the set $\{1, 2, \ldots, L\}$ and then randomly selecting $L - l$ locations in this permuted string and replacing the numbers in the corresponding locations by 0's. The non-zero numbers are $p_1, p_2, \ldots, p_l$. Each q_i^* is either null or a substring of 0's and $|q_0^*| + |q_1^*| + \cdots + |q_l^*| = L - l$.

$\mathsf{Insert}\,(P, a_1 a_2 \ldots a_l)$: Given $P = q_0^* p_1 q_1^* p_2 q_2^* \cdots p_l q_l^*$ and an l-bit binary string $a_1 a_2 \ldots a_l$, outputs a binary string $b_0^* a_1 b_1^* a_2 b_2^* \cdots a_l b_l^*$, where each b_i^* is a random binary substring whose length is equal to q_i^*.

Note. Notice that in this procedure the human user has to input random bits at the positions of q_i^*'s. This is an idealized assumption since humans may not be able to generate truly random bits. We acknowledge this as a drawback but use it nevertheless since it makes our analysis simpler.

$\mathsf{Sep}\,(P, r)$: From the given $P = q_0^* p_1 q_1^* p_2 q_2^* \cdots p_l q_l^*$ and an L-bit binary string $r = r_1 r_2 \ldots r_L$, does the following:

- $r' \leftarrow null$
- For $k = 1 \text{to} L$
  - If $P[k] \neq 0$
    - $r' \leftarrow r' || r\,[k]$
    - Output r'

Notice that, $|r'| = l$.

We are now ready to describe our protocol. After presenting the protocol based on a generic building block F and defining the security of our protocol under the assumed hardness of this building block, we will describe a suitable instantiation of this building block in the next section. It is our thesis that the proposed candidate satisfies our security requirements.

System Parameters. L, l and m; all positive integers.
Setup. H and C evaluate $\mathsf{Perm}(L, l)$ and keep the resulting hidden permutation P as a shared secret. From a family of (δ, τ)-hard AI problem solvers $F : \mathsf{Keys}(F) \times S \to \{0, 1\}$, H and C randomly select a secret key $k \in \mathsf{Keys}(F)$. C also sets $S^- = \{\}$.
Protocol.

- Set $j = 0$
- While $j \neq m$ or $j \neq \perp$:
  - C randomly chooses a binary sequence $b_1 b_2 \cdots b_L$ from $\{0, 1\}^L$.
  - for $i = 1$ to L:
    - If $b_i = 1$, C selects a random $s_i \in S - S^-$ such that $F_k(s_i) = 1$ and updates $S^- \leftarrow S^- \cup \{s_i\}$.
    - Else C selects a random $s_i \in S - S^-$ such that $F_k(s_i) = 0$ and updates $S^- \leftarrow S^- \cup \{s_i\}$.
    - Csets $s = s_1 s_2 \cdots s_L$ and sends it to H.
  - For $i = 1$ to l:
    - H computes $F(s_{p_i})$. If it is 1, it assigns $a_i = 1$ otherwise assigns $a_i = 0$.
  - H sends $r' = \mathsf{Insert}(P, a_1 a_2 \ldots a_l)$ to C.
  - If $\mathsf{Sep}(P, r') = F(s_{p_1}) \parallel F(s_{p_2}) \parallel \ldots \parallel F(s_{p_l})$, C increments j otherwise C sets $j = \perp$.
- If $j = m$, C accepts H.

It is easy to see that the probability of someone impersonating a legitimate user by randomly submitting answers is 2^{-lm}. We have defined the set S^-, so that once an input from a set S has been used, it should no longer be used again for that particular user. In practice we can define two sets: S_1 and S_0 denoting the sets whose elements evaluate to $F_k(.) = 1$ and 0 respectively. Each time an input is used from this set, it is taken out of this set and never used for the same user. The reason and practicality of this requirement will become clear when we show a reasonable instantiation.

5 Security Analysis

5.1 Security Against Passive Adversarial Programs

An Identification Protocol (H, C) is (p, q) secure against passive adversaries if for all computationally bounded adversaries $\mathcal{A}$,

$$\Pr\left[\langle \mathcal{A}\left(T^q\left(H(z), C(z)\right)\right), C(z)\rangle = accept\right] \leq p$$

Here $T^q(.,.)$ represents the transcript of q communication sessions between H and C. In the appendix, we assume that if $\mathcal{B}$ is a program, then even after observing a certain amount of values of $F_k(.)$, it cannot solve this function with probability better than δ. With this assumption, we prove the following for our protocol:

Claim 1. If F is a (δ, τ)-hard family of AI problem solvers, then for all adversarial programs $\mathcal{A}$:

$$\Pr\left[\left(\mathcal{A}\left(T^{q}\left(H\left(z\right), C\left(z\right)\right)\right), C\left(z\right)\right) = accept\right] \leq 2\delta - 1$$

Proof. See Appendix A.1.

5.2 Security Against Passive Human Adversaries

For a human adversary, we assume that F is a family of $(\lambda\left(r\right), \tau)$-resilient functions. Hence it is not possible to extract the key k of a particular function F_k of this family, when less than r of the instances are shown to a human adversary. We can construct an experiment in which this adversary is also given a random binary sequence of r bits. Obviously, this adversary has no advantage in detecting the key k, if this sequence is truly independent of the choice of answers. However if we give this adversary the true sequence of answers, then it might have considerable advantage in detecting the secret key. All we need to show is that our protocol does not reveal the true answers with all but a small probability. Ideally, the answer string should be a pseudorandom binary string; however we do not know how a human would be able to generate a cryptographically secure pseudorandom string without any computational aid. The best we can do is to introduce some randomness in the answer strings based on the secret permutation and the random bits. We digress here to explain the need for secret permutation and the random bits. Consider an adversary that randomly picks up a permutation and the corresponding locations and tries to detect the key. For a suitable choice of $L = 10$ and $l = 5$, the probability that the adversary's guess was correct is $\frac{5!}{10!}/\binom{10}{5} \approx 2^{-22}$. If instead, we use a simpler procedure of randomly selecting 5 locations (as in Section 2), then the adversary's probability of guessing the hidden locations is just $1/\binom{10}{5} \approx 2^{-8}$. To achieve the same level of probability, we would have to choose $L = 24$ and $l = 12$, which means that the user has to remember more locations. This motivates the use of our procedure.

We show in Appendix A.2, that in a given round, the probability of the event that the answer string returned by H is equal to the actual answer string (without permutation and random bits) is less than $\frac{1}{2^r}\left(1 + \frac{l}{L^2}\right)^r$. Since our adversary has no advantage in solving our protocol without guessing the random permutation, it follows that the advantage of this adversary in breaking our protocol is bounded by $\lambda\left(r\right)$ and the above probability, as shown in the appendix.

5.3 Security Against Active Adversaries

We take the definition of an active adversary from [5]. We do not distinguish between a human adversary and an adversarial program in this case. *An identification protocol (H, C) is (p, q, r)-detecting against active adversaries if for all computationally bounded adversaries $\mathcal{A}$,*

$$-\Pr\left[\left\langle H\left(z\right), \mathcal{A}\left(T^{r}\left(H\left(z\right), C\left(z\right)\right)\right)\right\rangle \neq \perp\right] < q$$
$$-\Pr\left[\left\langle \mathcal{A}\left(T^{r}\left(H\left(z\right), C\left(z\right)\right)\right), C\left(z\right)\right\rangle = accept\right] < p$$

Against active adversaries we have a natural defense thanks to the cognitive abilities of humans. If a human can detect that a particular instance has been replayed twice or more with a high probability, then he may reject C, hence terminating the protocol session. Hence our protocol has an innate defense against replay attacks. More specifically, let $1-q$ be the probability that H can detect an instance s' being replayed. Obviously, this probability should be a function of time and the specific iteration at which the instance is being replayed. Let S_i^- denote the variable defining the set S^- after the ith query to C by H. Let A_i be the event that an s' was presented to H at the ith query, such that $s' \in S_j$ for some $j < i$. Then,

$$\Pr\left[H \text{ detects } A_i\right] \geq \Pr\left[H \text{ detects } A_r | s' \in S_1^- \wedge s' \notin S_r^-, 1 < j < r\right] \geq 1 - q$$

All other events will be predicted with greater probability. Thus with high probability the human will detect a replay challenge attack.

Notice however, that our protocol is not secure against another kind of attack: the automated adversary intercepts the sequence from the server to the user, and replaces one instance in that sequence by some other instance (taken from some other source). If the user's reply is rejected, the adversary now has two instances for which it knows that the answers are different. After collecting a few such pairs, all these pairs are displayed to a human adversary, which now has the instances alongwith their answers. This attack, however would result in the termination of the session, if successful. The protocol could be repaired to handle this kind of active attack. Namely, we can allow the user to submit wrong answers about half of the time. This would result in an increase in the number of rounds and would take more time. We acknowledge it as a weakness in our protocol and leave a possible efficient fix as a future work. There may be other active attacks from a combined human-computer adversary and a thorough analysis is certainly required.

6 A Suitable Instantiation of F

We describe a procedure that seems a plausible candidate for an AI problem solver as well as being $(\lambda(r), \tau)$-resilient at the same time. Our motivation is the saying that a picture is worth a thousand words. A picture might satisfy a variety of predicates. Consider as an example, the picture in Figure 1. How many different predicates does this picture satisfy? To list a few:

- Does this picture present a cartoon?
- Is there a "nose" in this picture?
- Is there a woman in this picture?

And so on. In short, we can select a predicate, find a set of pictures that satisfy this predicate and a set that does not. We present these pictures to a human user in place of F in our protocol and we are sure that it will satisfy our goals. Let Pic denote a collection of pictures and let *pic* be a member of it. Let Pred denote the set of all predicates. We define the family of functions:

$$Q : \mathsf{Pred} \times \mathsf{Pic} \rightarrow \{0, 1\}$$

Fig. 1. A picture might represent a large number of concepts and contexts

Conjecture 1. Q defines a family of AI problem solvers.

The CAPTCHA project has a particular CAPTCHA named ESP-PIX [11], that presents a set of pictures related to each other through some feature. The pictures are rather obviously related to each other when viewed by a human; however it is claimed that for a computer program it is extremely hard to find a common link between the pictures. We present a similar but harder problem. In our protocol, the pictures satisfying the predicate are intermingled with those that do not, and we ask a computer program to tell whether the pictures are interlinked or not.

Conjecture 2. Q defines an $(\lambda(r), \tau)$-resilient family of functions.

This claim seems hard to justify. Our claim is based on the belief that the following problem would be hard for a human. Namely, a human is given a series of pictures as in Figure 2; is told that some of these pictures satisfy a given predicate and some don't, and is asked to guess the hidden predicate. Of course without the knowledge of the answers to the predicate for each of these pictures, this seems to be a hard problem. With the knowledge of the answers, we cannot

Fig. 2. A set of pictures, some or all of which satisfy a hidden predicate

say with a certain amount of confidence that a person might not be able to guess the hidden predicate. We constructed our protocol in a way so as to hide the answers, such that the adversary might not be able to gain advantage by guessing the hidden predicate. An important question is: What kind of predicate to choose? A predicate involving color differences like "Does the picture contain

the color red?" should most certainly not be used. Color difference is very easily caught by the human eye. The predicate used in Figure 2 is "Does the object in the image begin contain the letter "P" in its name?" Predicates like this, which do not catch the human eye, are the likely candidates.

6.1 A User Friendly Implementation

For human users the parameters $L = 10, l = 5$ and $m = 4$ can be chosen. Note that a random guess attack can only succeed with a probability 2^{-20} in this case, which is more than the security of a 4-digit pin number. The user is given a hidden permutation string say: 0098030502. When the user inputs his ID, he is brought to a page containing 10 pictures in a 2X5 grid at the bottom of which is a text box. The user answers by randomly picking '0' or '1' in place of the '0's' in the permutation string and answering the pictures in the specified order corresponding to the digits other than the '0's'. So, for example, a possible answer would be 10<u>1</u>10<u>0</u>1<u>1</u>01, where the underlined digits denote the actual answers and the rest are random bits. The user would input the string 1011001101 and will go to the next series of 10 pictures if this answer is correct at the specified positions. The procedure continues until 4 steps and the user is accepted once all the 4 steps result in a success. The choice of using 10 pictures in a challenge seems appropriate, as they can easily be displayed on the screen as shown in Figure 3. A single picture would take around 5 seconds for a human

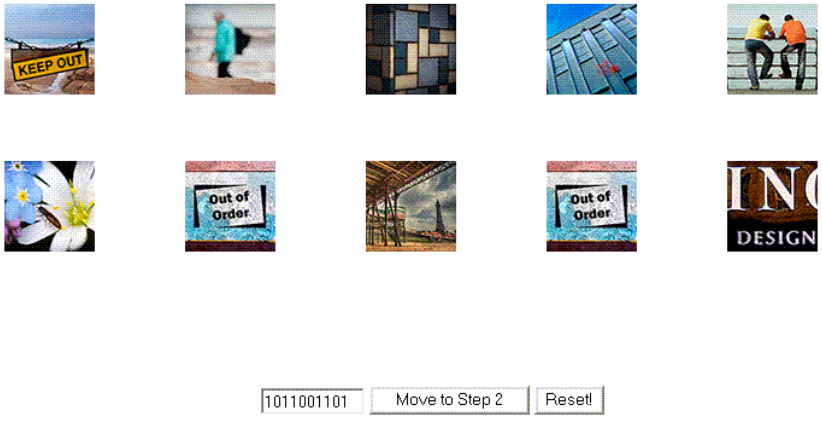

Fig. 3. An authentication step in our proposed protocol

to verify whether it satisfies the predicate or not. This would take around 100 seconds to execute the protocol. This amount of time seems reasonable if we use the protocol only under certain circumstances such as when the user is trying to log on through an insecure public terminal.

7 Limitations and Discussion

One might ask the question that how long the protocol should be run to keep a desired security level. It is evident that based on our assumption, the protocol can be safely executed for a number of times if we only consider adversarial programs. How about human adversaries? We know an inherent weakness in us humans; the more the work load, the less efficient we are. So if a human adversary is given a collection of, say 2000, pictures and is asked to find the hidden predicate, then he might not be able to examine all these pictures. In order to make the task harder for a human adversary, we can have two or more predicates connected through a truth clause. The user then checks whether the picture satisfies the clause and answers accordingly. This will result in an increase in execution time, but guessing the secret predicates will be much harder.

Another question is regarding the possible usage of our protocol. We insist that our protocol should be used only in situations when a user is away from the luxury and security of his personal computer or office environment. The user might want to use our system when using a public terminal to log in for emails while on a business trip. However, when he is back in his office or home, he can use the normal password based system to log in to his computer. So, we can safely use our system for a small number of authentications before the secret predicate can be refreshed and a new hidden permutation can be used. The fact that the secret predicate plus the hidden permutation is not a load on a human's memory (the hidden permutation being the size of a normal telephone number) makes this switch very practical and reusable. Consequently, we can use this system, for say 20 or 30 authentications before renewing the secret.

A modified version of the protocol, secure against general active adversaries is also desirable; without making it infeasible or increasing the number of rounds. The most important limitation is the selection of the predicates and selecting appropriate pictures that satisfy these predicates. This can be done by a dedicated group from the service providers. We do not know however, if this task can be performed by a computer or not. Automatically generated predicates and pictures might prove helpful and will increase the practicality of our scheme. Another important direction is to find whether there exist other functions in place of the predicate-image one. A function family whose soundness can be theoretically proved instead of being conjectured would certainly be a better candidate than the one presented in this paper.

8 Conclusion

The problem of making secure human identification protocols in which a human authenticates to a remote server has resulted in many efficient authentication protocols over the years. These protocols try to make things easy for humans by presenting them a challenge based on some mathematical problem which is easy to compute but difficult for an adversary to crack. However, the efficiency of these protocols lie in the user friendly representation. Instead of using computational problems, we can look for problems in domains where humans are better

than computers, like image evaluation. However, to construct an identification protocol, we need some problem that is not only hard for computer programs but for human adversaries too. We have shown that it is possible to construct a protocol based on human's excellent image processing abilities such that defeating the protocol is hard even for the human adversaries. The proposed problem based on evaluating an image through a secret predicate seems to be hard to crack even for human adversaries who do not have the knowledge of the secret. This allows us to deal with the security of the protocol separately for the adversarial programs and for human adversaries. It will be interesting to investigate further in search for other possible problems that satisfy this nice feature. The obvious open question and limitation is to mathematically quantify the hardness of the problem discussed in this paper. This, however, remains an open problem.

Acknowledgements

We are grateful to Stuart Haber for his comments and the anonymous reviewers for their suggestions.

References

1. Matsumoto, T., Imai, H.: Human Identification through Insecure Channel. Advances in Cryptology - EUROCRYPT 91, Lecture Notes in Computer Science, Springer-Verlag. **547** (1991) 409–421
2. Wang, C.H., Hwang, T., Tsai, J.J.: On the Matsumoto and Imai's Human Identification Scheme. Advances in Cryptology - EUROCRYPT 95, Lecture Notes in Computer Science, Springer-Verlag. **921** (1995) 382–392
3. Matsumoto, T.: Human-computer cryptography: An attempt. 3rd ACM Conference on Computer and Communications Security, ACM Press. (1996) 68–75
4. Xiang-Yang Li, Shang-Hua Teng: Practical Human-Machine Identification over Insecure Channels. Journal of Combinatorial Optimization. **3** (1999) 347–361
5. Hopper, N.J., Blum, M.: Secure Human Identification Protocols. Advances in Cryptology - Asiacrypt 2001, Lecture Notes in Computer Science, Springer-Verlag. **2248** (2001) 52–66
6. Luis von Ahn, Manuel Blum, Nicholas Hopper, John Langford: CAPTCHA: Using Hard AI Problems for Security. Advances in Cryptology – Eurocrypt 2003, Lecture Notes in Computer Science, Springer-Verlag. (2003) 294–311
7. Shujun Li, Heung-Yeung Shum: Secure Human-computer identification against Peeping Attacks (SecHCI): A Survey. Unpublished report, available at Elsevier's Computer Science Preprint Server. (2002)
8. Rachna Dhamija, Adrian Perrig: Deja Vu: A user study using images for authentication. Proc. of the 9th USENIX Security Symposium. (2000) 45–58
9. ID Arts Inc: Passfaces - the Art of Identification. Visit http://www.idarts.com
10. Vince Sorensen: PassPic - Visual Password Management. Visit http://www.authord.com/
11. The CAPTCHA project: ESP-PIX. Visit http://www.captcha.net/
12. Daphna Weinshall: Cognitive Authentication Schemes Safe Against Spyware (Short Paper). 2006 IEEE Symposium on Security and Privacy. (2006) 295–300
13. Philippe Golle and David Wagner: Cryptanalysis of a Cognitive Authentication Scheme. Cryptology ePrint Archive, Report 2006/258. http://eprint.iacr.org/.

A Security Analysis

A.1 Security Against Passive Adversarial Programs

We first define the following oracles that represent the different functionalities in our protocol.

The oracle F_k. For any $k \in \mathsf{Keys}(F)$, this oracle takes as input an $s \in S$ and outputs $F_k(s) \in \{0,1\}$.

We assume a global set S^- initially empty, available to the following oracles.

The oracle C. This oracle takes as input the following queries:

- Init : This query initializes a new session and terminates any previous session. C randomly outputs the sequence of instances $s = s_1 s_2 \dots s_L$ from $S - S^-$ and updates $S^- \leftarrow S^- \cup s$, $j \leftarrow 1$.
- $C(a_1 a_2 \dots a_L)$: If $j = \perp$ outputs *reject*. Else if $j = m$ outputs *accept* or *reject* and updates $j \leftarrow \perp$. Else if $j < m$, yields one of two possible outputs:
  - (s) ; Randomly outputs the sequence $s = s_1 s_2 \dots s_L$ from $S - S^-$ and updates $S^- \leftarrow S^- \cup s$, $j \leftarrow j + 1$.
  - $(reject)$; Outputs $(reject)$ and sets $j \leftarrow \perp$.

The oracle H. Takes as input the query $H(s = s_1 s_2 \dots s_L)$. If $S^- \cap s = \varphi$, outputs $a_1 a_2 \dots a_l$. Else outputs $\perp$.

Now suppose an AI problem solver outputs a sequence of bits $r_1 r_2 \dots r_t$ on the inputs $s_1 s_2 \dots s_t$. An adversarial program $\mathcal{A}$ wants to guess $F_k(s_{t+1})$ on being given the challenge s_{t+1}. The following experiment describes this functionality:

Experiment $\mathbf{Exp}^{aps}_{F,\mathcal{A}}$
$k \xleftarrow{R} \mathsf{Keys}(F)$
$s \leftarrow \mathcal{A}^{F_k}$
$b \leftarrow \mathcal{A}^{F_k}(s)$
If $b = F_k(s)$ return 1 else return 0

The *aps-advantage* of $\mathcal{A}$ is defined as:

$$\mathbf{Adv}^{aps}_{F,\mathcal{A}} = \Pr\left[\mathbf{Exp}^{aps}_{F,\mathcal{A}} = 1\right]$$

For any t, q we define the *aps-advantage* of F as:

$$\mathbf{Adv}^{aps}_{F,\mathcal{A}}(t,q) = \max_{\mathcal{A}}\left\{\mathbf{Adv}^{aps}_{F,\mathcal{A}}\right\}$$

with the maximum being over all adversarial programs $\mathcal{A}$ having time-complexity t and making at most q oracle queries to the oracle F_k.

Conjecture. If F is a family of (δ, τ)-hard AI problem solvers, then for any program $\mathcal{A}$:

$$\mathbf{Adv}_{F,\mathcal{A}}^{aps}(\tau, |S| - 1) < \delta.$$

where $\delta > 1/2$.

Our belief on this conjecture is based on the definition of a CAPTCHA[6]. Even if we provide answers to the queries of an adversarial program, it will be hard for it to analyze all the pictures and categorize them into one category. The function family Q described in Section 6 shows just this.

Now let $\mathcal{B}$ be a passive adversarial program, such that:

$$\Pr\left[\left(\mathcal{B}\left(T^{q}\left(H\left(z\right), C\left(z\right)\right)\right), C\left(z\right)\right) = accept\right] > \beta.$$

This adversary can be described by the following experiment:

Experiment $\mathbf{Exp}_{F,\mathcal{B}}^{aut}$
  Initialize oracles C and H
  While state = "test"
    done $\leftarrow \mathcal{B}^{C,H}$
  $\mathcal{B}$ queries C until C outputs *accept* or *reject*.
  If C outputs *accept*
    Output '1'
  Else
    Output '0'

We define the advantage of $\mathcal{B}$ as:

$$\mathbf{Adv}_{F,\mathcal{B}}^{aut} = \Pr\left[\mathbf{Exp}_{F,\mathcal{B}}^{aut} = 1\right]$$

And *aut-advantage* of our protocol as:

$$\mathbf{Adv}_{F,\mathcal{B}}^{aut}(t, q_C, q_H) = \max_{\mathcal{B}}\left\{\mathbf{Adv}_{F,\mathcal{B}}^{aut}\right\}$$

Now, we relate the two adversaries with the help of the following claim:

Claim. We have:

$$2\mathbf{Adv}_{F,\mathcal{A}}^{aps}(t_p + t_s q_C + t_a q_H + \tau, (q_C + q_H)\, l) - 1 = \mathbf{Adv}_{F,\mathcal{B}}^{aut}(\tau, q_C, q_H)$$

Proof. We construct an adversary $\mathcal{A}_{\mathcal{B}}$ which is given an oracle F_k and runs adversary $\mathcal{B}$ as a subroutine. This adversary uses the advantage of $\mathcal{B}$ in defeating our protocol to predict the image of F_k at s.

The adversary is described as follows:

Adversary $\mathcal{A}_{\mathcal{B}}^{F_k}$
  Randomly select $P = \mathsf{Perm}(L, l)$. Run Adversary $\mathcal{B}$, replying to its oracle queries as follows:
  When $\mathcal{B}$ makes an oracle query init:

Set $j \leftarrow 1$ and randomly select $s = s_1 s_2 \cdots s_L$, where $s_i \in S - S^-$ and update $S^- \leftarrow S^- \cup \{s_i\}$. Update the sequence $p \leftarrow s_{p_1} s_{p_2} \ldots s_{p_l}$.

When $\mathcal{B}$ makes an oracle query $C\,(r' = a_1 a_2 \ldots a_L)$, do:

If $j = \perp$ output *reject*.

Else if $j = m$ and $\mathsf{Sep}\,(P, r') = F\,(s_{p_1})\,F\,(s_{p_2}) \ldots F\,(s_{p_l})$ output *accept* and update $j \leftarrow \perp$; else *reject* and update $j \leftarrow \perp$.

Else if $j < m$, yields one of two possible outputs:

(s); If $\mathsf{Sep}\,(P, r') = F\,(s_{p_1})\,F\,(s_{p_2}) \ldots F\,(s_{p_l})$, randomly output the sequence $s = s_1 s_2 \ldots s_L$ from $S - S^-$ and update $S^- \leftarrow S^- \cup s$, $j \leftarrow j + 1$ and $p \leftarrow s_{p_1} s_{p_2} \ldots s_{p_l}$.

("*reject*"); If $\mathsf{Sep}\,(\mathsf{Perm}, r') \neq F\,(s_{p_1})\,F\,(s_{p_2}) \ldots F\,(s_{p_l})$ output "*reject*" and set $j \leftarrow \perp$.

When $\mathcal{B}$ makes an $H\,(s = s_1 s_2 \ldots s_L)$ query do:

For $i = 1$ to l, $a_i \leftarrow F_k\,(s_{p_i})$

Return $r' = \mathsf{Insert}\,(P, a_1 a_2 \ldots a_l)$ to $\mathcal{B}$ as the answer.

Until $\mathcal{B}$ outputs the state 'done'.

Set $j \leftarrow 0$.

On $\mathcal{B}$'s init query, randomly select $s = s_1 s_2 \cdots s_L$, where $s_i \in S - S^-$ and update $S^- \leftarrow S^- \cup \{s_i\}$, $j \leftarrow 1$. Update the sequence $p \leftarrow s_{p_1} s_{p_2} \ldots s_{p_l}$.

For 2 to m do:

When $\mathcal{B}$ outputs $r' = a_1 a_2 \ldots a_L$ set $q \leftarrow q || \mathsf{Sep}\,(P, r')$. Randomly select $s = s_1 s_2 \cdots s_L$, where $s_i \in S - S^-$ and update $S^- \leftarrow S^- \cup \{s_i\}$, $j \leftarrow j + 1$. Update the sequence $p \leftarrow p || s_{p_1} s_{p_2} \ldots s_{p_l}$.

When $\mathcal{B}$ outputs $a_1 a_2 \ldots a_L$, set $q \leftarrow q || \mathsf{Sep}\,(P, r')$ and output *accept*.

Randomly select an s from $p = s_1 s_2 \ldots s_{lm}$ and set it as the output.

Output the corresponding bit in $q = a_1 a_2 \ldots a_{lm}$ as the response and halt.

Now, we can see that:

$$\begin{aligned}
\mathbf{Adv}^{aps}_{F,\mathcal{A}_\mathcal{B}} &= \Pr_{r \in F_k}\left[s \leftarrow \mathcal{A}_\mathcal{B}^{F_k}; b \leftarrow \mathcal{A}_\mathcal{B}^{F_k}\,(s)\,; F_k\,(s) = b\right] \\
&= \Pr\left[F_k\,(s) = b | \mathcal{B}\,\text{succeeds}\right] \Pr\left[\,\mathcal{B}\,\text{succeeds}\right] \\
&\quad + \Pr\left[F_k\,(s) \neq b |\,\mathcal{B}\,\text{fails}\right] \Pr\left[\mathcal{B}\,\text{fails}\right] \\
&= \frac{\mathbf{Adv}^{aut}_{F,\mathcal{B}}}{2} + \frac{1}{2}
\end{aligned}$$

Let t_p, t_s and t_a denote the running times of the procedures $\mathsf{Perm}\,(.,.)$, $\mathsf{Sep}\,(.,.)$ and $\mathsf{Insert}\,(.,.)$ respectively. $\mathcal{A}_\mathcal{B}$ has to perform $\mathsf{Perm}\,(.,.)$ once, $\mathsf{Sep}\,(.,.)$ a maximum of $t_s q_C$ times and $\mathsf{Insert}\,(.,.)$ a maximum of $t_a q_H$ times. Further notice that apart from these calculations, the adversary $\mathcal{A}_\mathcal{B}$ does some rather trivial calculations more than the adversary $\mathcal{B}$. Thus if the running time of $\mathcal{B}$ is bounded by τ, then that of $\mathcal{A}_\mathcal{B}$ is bounded above by $t_p + t_s q_C + t_a q_H + \tau$. Hence, by maximizing, we reach to the conclusion:

$$2\mathbf{Adv}^{aps}_{F,\mathcal{A}}\,(t_p + t_s q_C + t_a q_H + \tau, (q_C + q_H)\,l) - 1 = \mathbf{Adv}^{aut}_{F,\mathcal{B}}\,(\tau, q_C, q_H)$$

□

Theorem. If F is a (δ, τ)-hard family of AI problem solvers, then for all passive adversarial programs $\mathcal{B}$ running in time less than $\tau - \Delta t$:

$$Pr\left[\left(\mathcal{B}\left(T^{q}\left(H\left(z\right), C\left(z\right)\right)\right), C\left(z\right)\right) = accept\right] < 2\delta - 1$$

where, $q \approx \frac{|S|-1}{Lm}$ and $\Delta t \approx \left(1 + \frac{|S|-1}{l}\right) t$, with $t \ll \tau$.

A.2 Security Against Passive Human Adversaries

Assume that we have a human adversary $\mathcal{H}$. We consider the following experiment:

Experiment $\mathbf{Exp}^{hg}_{F,\mathcal{H}}$
$k \overset{R}{\leftarrow} \mathsf{Keys}(F)$;
$b_1 b_2 \ldots b_k \leftarrow \{0,1\}^k$;
$s_1 s_2 \ldots s_r \leftarrow S - S^{-}$, such that $F\left(s_1\right) F\left(s_2\right) \ldots F\left(s_r\right) = b_1 b_2 \ldots b_k$;
$k' \leftarrow \mathcal{H}\left(s_1 s_2 \ldots s_r\right)$;
If $k = k'$ return 1 else return 0.

The *hg-advantage* of $\mathcal{H}$ is defined as:

$$\mathbf{Adv}^{hg}_{F,\mathcal{H}} = \Pr_{r \in F_k}\left[\mathbf{Exp}^{hg}_{F,\mathcal{A}} = 1\right]$$

For any t we define the *hg-advantage* of F as:

$$\mathbf{Adv}^{hg}_{F,\mathcal{H}}\left(t\right) = \max_{\mathcal{H}}\left\{\mathbf{Adv}^{hg}_{F,\mathcal{H}}\right\}$$

Since F is an $(\lambda\left(r\right), \tau)$-resilient function family, we have

$$\mathbf{Adv}^{hg}_{F,\mathcal{H}}\left(\tau\right) = \lambda\left(r\right).$$

This tells us that if a human adversary is given a series of instances, and is not shown which one of them output 1 or which one of them output zero, then he has a probability of $\lambda\left(r\right)$ in successfully guessing the key. All we to show is that for a human observer, the following two situations are hard to distinguish but with a small probability: (a) Concatenated outputs of the two oracles in our protocol for a total of r/Lm authentication sessions. (b) A random set of r instances with a truly random answer bit string of the same length. In particular, consider a passive adversary $\mathcal{H}$, who listens r/Lm sessions of our protocol and gets the r instances $s_1 s_2 \ldots s_r$ together with their answers $a_1 a_2 \ldots a_r$. Let A_i denote the event that $F\left(s_i\right) = a_i$ in our protocol. We first prove the following:

Theorem. $\frac{1}{2^r} < \Pr\left[A_1 \wedge A_2 \wedge \ldots \wedge A_r\right] < \frac{1}{2^r}\left(1 + \frac{l}{L^2}\right)^r$, where $r = jL$ for some positive integer j.

Proof. Consider the event, $A_1 : F(s_1) = a_1$. Let σ denote the identity permutation and P denote the hidden permutation of our protocol, then,

$$\begin{aligned}\Pr[F(s_1) = a_1] &= \Pr[F(s_1) = a_1 | \sigma(1) = P(1)] \Pr[\sigma(1) = P(1)] + \\ &+ \Pr[F(s_1) = a_1 | \sigma(1) \neq P(1)] \Pr[\sigma(1) \neq P(1)] \\ &= 1 \cdot \frac{1}{L}\left(\frac{l}{L}\right) + \frac{1}{2}\left(1 - \frac{l}{L^2}\right) = \frac{1}{2}\left(1 + \frac{l}{L^2}\right)\end{aligned}$$

Similarly, we can find out that:

$$\Pr[F(s_2) = a_2 | F(s_1) = a_1] < \frac{1}{2}\left(1 + \frac{l}{L^2}\right)$$

And in general, for all $t \leq r$,

$$\frac{1}{2} < \Pr[F(s_t) = a_t | F(s_1) = a_1 \wedge \ldots \wedge F(s_{t-1}) = a_{t-1}] < \frac{1}{2}\left(1 + \frac{l}{L^2}\right)$$

Hence, $\frac{1}{2^r} < \Pr[A_1 \wedge A_2 \wedge \ldots \wedge A_r] < \frac{1}{2^r}\left(1 + \frac{l}{L^2}\right)^r$. □

We define the advantage of a passive human adversary attempting to defeat our protocol as $\mathbf{Adv}^{hg}_{P,\mathcal{H}}(\tau)$ and assume that, $\mathbf{Adv}^{hg}_{P,\mathcal{H}}(\tau) = \mathbf{Adv}^{hg}_{F,\mathcal{H}}(\tau) + \alpha(r)$; where, $\alpha(r) = M$,$(M < 1 - \lambda(r))$ when $\Pr[A_1 \wedge A_2 \wedge \ldots \wedge A_r] = 1$, and $\alpha(r) = 0$ when, $\Pr[A_1 \wedge A_2 \wedge \ldots \wedge A_r] = 1/2^r$. Assuming $\alpha(r)$, to be a linear function, it is straight forward to show that:

$$\mathbf{Adv}^{hg}_{P,\mathcal{H}}(\tau) < \lambda(r) + \frac{M}{2^r - 1}\left(\left(1 + \frac{l}{L^2}\right)^r - 1\right).$$

Cryptanalysis of Reduced Variants of the FORK-256 Hash Function*

Florian Mendel[1,**], Joseph Lano[2], and Bart Preneel[2]

[1] Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria
Florian.Mendel@iaik.tugraz.at
[2] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium
{Joseph.Lano,Bart.Preneel}@esat.kuleuven.be

Abstract. FORK-256 is a hash function presented at FSE 2006. Whereas SHA-like designs process messages in one stream, FORK-256 uses four parallel streams for hashing. In this article, we present the first cryptanalytic results on this design strategy. First, we study a linearized variant of FORK-256, and show several unusual properties of this linearized variant. We also explain why the linearized model can not be used to mount attacks similar to the recent attacks by Wang *et al.* on SHA-like hash functions. Second, we show how collision attacks, exploiting the non-bijectiveness of the nonlinear functions of FORK-256, can be mounted on reduced variants of FORK-256. We show an efficient attack on FORK-256 reduced to 2 streams and present actual colliding pairs. We expect that our attack can also be extended to FORK-256 reduced to 3 streams. For the moment our approach does not appear to be applicable to the full FORK-256 hash function.

1 Introduction

Recent results in cryptanalysis of hash functions [6,5] show weaknesses in many commonly used hash functions, such as SHA-1 and MD5. Therefore, the cryptanalysis of alternative hash functions is of great interest. In this article, we will study the hash function FORK-256. It was proposed by Hong *et al.* at FSE 2006 [2]. FORK-256 was designed to be resistant against known-attack strategies including the attack by Wang *et al.* used to break SHA-1 [5].

In this article, we present the first cryptanalytic results on FORK-256 and stream-reduced variants. On the one hand we explain why the linearized model can not be used to mount attacks similar to the attack of Wang *et al.* on SHA-1. All the characteristics we found in the linearized variant of the hash function have a low probability to hold in the original FORK-256 hash function. On the

* This previous work was in part supported by grant No.2005-S-062(2005) from the KISA(Korea Information Security Agency).
** This author is supported by the Austrian Science Fund (FWF), project P18138.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 85–100, 2007.

other hand, we show several unusual properties in the linearized variant of the hash function, which are not common in the linearized variants of other hash functions, as for instance the SHA-family [3].

Furthermore, we show how collision attacks, exploiting the non-bijectiveness of the nonlinear functions of FORK-256, can be mounted on reduced variants of FORK-256. We show an efficient attack on FORK-256 reduced to 2 streams and present a colliding message pair. We expect that the attack can be extended to FORK-256 reduced to 3 streams.

The remainder of this article is structured as follows. A description of the hash function is given in Section 2. In Section 3, we show that the linearized variant of the FORK-256 has several unusual properties. Differential attacks on FORK-256 using the linearized variant for finding a suitable characteristic are studied in Section 4. In Section 5, we give a truncated differential which can be used to break stream-reduced variants of FORK-256. A sample colliding message pair for FORK-256 reduced to two streams is given in this section as well. Conclusions are presented in Section 6.

2 Description of the Hash Function FORK-256

The FORK-256 hash function was proposed by Hong *et al.* in [2]. It is an iterative hash function that processes 512-bit input message blocks and produces a 256-bit hash value. Unlike other commonly used hash functions, such as the SHA-family, it consists of 4 parallel streams which we denote B_1, B_2, B_3 and B_4. In each stream the state variables are updated according to the expanded message words and combined with the chaining variables after the last step, depicted in Fig. 1. In the following, we briefly describe the FORK-256 hash function. It basically consists of two parts: the message expansion and the state update transformation. A detailed description of the hash function is given in [2].

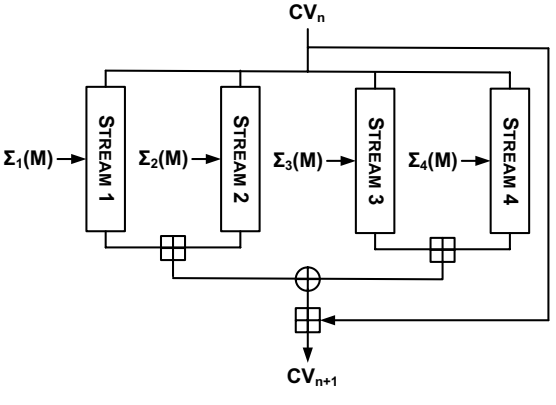

Fig. 1. Structure of FORK-256

2.1 Message Expansion

The message expansion of FORK-256 is a permutation of the 16 message words m_i in each stream, where different permutations are used. The ordering of the message words for each stream is given by the permutations Σ_1, Σ_2, Σ_3 and Σ_4, where $\Sigma_j(M) = w_j = (m_{\sigma_j(0)}, \ldots, m_{\sigma_j(15)})$.

Table 1. Ordering of the message words

step k	0		1		2		3		4		5		6		7	
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_1(i)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_2(i)$	14	15	11	9	8	10	3	4	2	13	0	5	6	7	12	1
$\sigma_3(i)$	7	6	10	14	13	2	9	12	11	4	15	8	5	0	1	3
$\sigma_4(i)$	5	12	1	8	15	0	13	11	3	10	9	2	7	14	4	6

2.2 State Update Transformation

The state update transformation starts from a (fixed) initial value IV of eight 32-bit registers and updates them in 4 parallel streams of 8 steps. In each step 2 message words are used to update the eight state variables. Fig. 2 shows one step of the state update transformation of FORK-256.

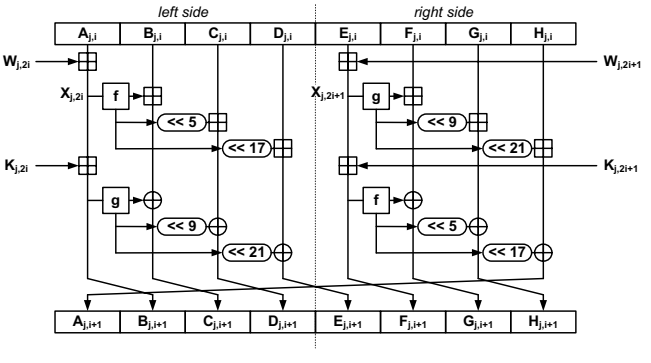

Fig. 2. Step i in stream B_j of FORK-256

The non-linear functions f and g used in each step are defined as follows.

$$f(x) = x + (x \lll 7 \oplus x \lll 22)$$
$$g(x) = x \oplus (x \lll 13 + x \lll 27)$$

Two step constants $K_{j,2i}$ and $K_{j,2i+1}$ are added in step i; the constants are different for each step of the stream. The order of the constants is different in

Table 2. Ordering of constants

step i	$K_{1,2i}$	$K_{1,2i+1}$	$K_{2,2i}$	$K_{2,2i+1}$	$K_{3,2i}$	$K_{3,2i+1}$	$K_{4,2i}$	$K_{4,2i+1}$
0	δ_0	δ_1	δ_{15}	δ_{14}	δ_1	δ_0	δ_{14}	δ_{15}
1	δ_2	δ_3	δ_{13}	δ_{12}	δ_3	δ_2	δ_{12}	δ_{13}
2	δ_4	δ_5	δ_{11}	δ_{10}	δ_5	δ_4	δ_{10}	δ_{11}
3	δ_6	δ_7	δ_9	δ_8	δ_7	δ_6	δ_8	δ_9
4	δ_8	δ_9	δ_7	δ_6	δ_9	δ_8	δ_6	δ_7
5	δ_{10}	δ_{11}	δ_5	δ_4	δ_{11}	δ_{10}	δ_4	δ_5
6	δ_{12}	δ_{13}	δ_3	δ_2	δ_{13}	δ_{12}	δ_2	δ_3
7	δ_{14}	δ_{15}	δ_1	δ_0	δ_{15}	δ_{14}	δ_0	δ_1

each stream. The ordering of the constants for each stream is given in Table 2. For the actual values of the constants δ_0 to δ_{15} we refer to [2].

After the last step of the state update transformation, the chaining variables and the output values of the last step of the four streams are combined, resulting in the final value of one iteration (feed forward). The feed forward is a word-wise modular addition of the IV and the output of the state update transformation. The result is the final hash value or the initial value for the next message block.

3 L-FORK-256: A Linearized Variant of FORK-256

In this section, we analyze the linearized variant of FORK-256. We show that the linearized variant L-FORK-256 has several properties that are not common in the linearized variants of other hash functions. However, so far we do not see how these properties can be used in an attack on the original FORK-256 hash function. L-FORK-256 is constructed by replacing all modular additions in the hash function by XOR operations.

$$LH(CV_n, M_n) = CV_n \oplus \bigoplus_{j=1}^{4} B_j(CV_n, \Sigma_j(M_n)) \tag{1}$$

The 4 streams B_j can be described as follows.

$$B_j(CV_n, \Sigma_j(M_n)) = CV_n\mathrm{A} \oplus \Sigma_j(M_n)\mathrm{B} \oplus \mathrm{c}_j \tag{2}$$

$$= CV_n\mathrm{A} \oplus M_n\mathrm{S}_j\mathrm{B} \oplus \mathrm{c}_j \tag{3}$$

with S_j some permutation matrices, and A, B matrices that describe the action of L-FORK-256 on the chaining value input, respectively the message input. The matrices A are of little importance, since:

$$LH(CV_n, M_n) = CV_n \oplus \bigoplus_{j=1}^{4} CV_n\mathrm{A} \oplus M_n\mathrm{S}_j\mathrm{B} \oplus \mathrm{c}_j \tag{4}$$

$$= CV_n \oplus M_n(\mathrm{S}_1 \oplus \mathrm{S}_2 \oplus \mathrm{S}_3 \oplus \mathrm{S}_4)\mathrm{B} \oplus (\mathrm{c}_1 \oplus \mathrm{c}_2 \oplus \mathrm{c}_3 \oplus \mathrm{c}_4) \tag{5}$$

$$= CV_n \oplus M_n\mathrm{SB} \oplus \mathrm{c}, \tag{6}$$

with $\mathrm{S} = \mathrm{S}_1 \oplus \mathrm{S}_2 \oplus \mathrm{S}_3 \oplus \mathrm{S}_4$ and $\mathrm{c} = \mathrm{c}_1 \oplus \mathrm{c}_2 \oplus \mathrm{c}_3 \oplus \mathrm{c}_4$.

3.1 Action on Special Message Words

The four streams of FORK-256 are quite similar. Only the ordering of the constants and the message words is different in each stream.

Observation 1. *If we have a message M with repeating message words $M = m_0, m_1, \ldots, m_{15}$ with $m_i = m_j \forall i, j$, then $\Sigma_1(M) = \Sigma_2(M) = \Sigma_3(M) = \Sigma_4(M)$.*

Observation 2. *In L-FORK-256, for inputs that are repeating messages, we have that $B_1 = B_2 = B_3 = B_4$.*

Consequently, the description (2) of a stream can be reduced to:

$$B_j(CV_n, M_n) = CV_n \mathrm{A} \oplus M_n \mathrm{B} \oplus \mathrm{c}_j \tag{7}$$

and the description(6) of L-FORK-256 becomes

$$LH(CV_n, M_n) = CV_n \oplus \mathrm{c}, \tag{8}$$

which is independent of M_n.

Observe that having a repeating message as input is a *sufficient* condition for this effect, but it is not a necessary condition. It can be verified that (8) holds whenever the input message satisfies the following 12 conditions:

$$\begin{aligned}
&m_6 = m_7 = m_9 \\
&m_3 = m_5 = m_{12} = m_{13} \\
&m_1 = m_2 = m_3 \oplus m_6 \oplus m_{15} \\
&m_0 = m_8 = m_3 \oplus m_6 \oplus m_{14} \\
&m_{10} = m_{11} = m_3 \oplus m_{14} \oplus m_{15} \\
&m_4 = m_6 \oplus m_{14} \oplus m_{15} \ .
\end{aligned} \tag{9}$$

3.2 Fixed-Points

In L-FORK-256 we can easily construct a fix-point for any value of the chaining variables CV_n.

Theorem 1. *A two-block message can be used to construct a two-step fixed point in L-FORK-256.*

Proof: By combining two repeated messages, we can construct a fixed point for L-FORK-256. Let $M_1 = m \| \cdots \| m$ and $M_2 = \overline{m} \| \cdots \| \overline{m}$, then

$$\begin{aligned}
CV_n &= LH(CV_{n-1}, M_2) \\
&= CV_{n-1} \oplus \mathrm{c} \\
&= LH(CV_{n-2}, M_1) \oplus \mathrm{c} \\
&= (CV_{n-2} \oplus \mathrm{c}) \oplus \mathrm{c} \\
&= CV_{n-2} \ .
\end{aligned}$$

□

Theorem 2. *Two fixed-points and an arbitrary message block M_3 can be used to produce a collision in L-FORK-256.*

Proof: Let M_1, M_2 be repeating messages and let M_3 be an arbitrary message. Define $y = \text{L-FORK-256}(IV, M_3)$. Then the hash values of $M = M_3|M_2|M_1$ and $M^* = M_2|M_1|M_3$ are given by:

$$\begin{aligned} CV_2 &= LH(IV, M_1) = \mathrm{c} \oplus IV \\ CV_3 &= LH(CV_2, M_2) = \mathrm{c} \oplus \mathrm{c} \oplus IV = IV \\ CV_4 &= LH(CV_3, M_3) = LH(IV, M_3) = y \end{aligned}$$

$$\begin{aligned} CV_2^* &= LH(IV, M_3) = y \\ CV_3^* &= LH(CV_2^*, M_1) = \mathrm{c} \oplus y \\ CV_4^* &= LH(CV_3^*, M_2) = \mathrm{c} \oplus \mathrm{c} \oplus y = y \end{aligned}$$

□

3.3 Output Dependencies

In L-FORK-256 we found several output dependencies. However, statistical tests show that these are not present in the original hash function.

Observation 3. *Three linear dependencies exist between the 256 output bits of L-FORK-256. These 3 dependencies are the following:*

$$\sum_{i=0}^{127} w_{2i+1} = 0, \quad \sum_{i=1}^{128} w_{2i} = 0, \quad \sum_{i=33}^{160} w_i = 0 \ . \tag{10}$$

From (10) it follows that the parity of the output of L-FORK-256 is constant.

4 Differential Analysis

In this section, we analyze the security of FORK-256 against differential attacks. We study the impact of the type of attack that was used by Wang *et al.* to break SHA-1 [5]. The attack can be summarized as follows. First, find a collision producing characteristic with high probability in the linearized variant of the hash function. Second, use random trials to find a message pair that follows the linear characteristic.

4.1 Finding a Characteristic

Finding a collision in the linearized variant of FORK-256 is not difficult since it depends only on the differences in the message words. Two messages M and $M^* = M \oplus \Delta$ collide if and only if:

$$h_1^* \oplus h_1 = (\mathrm{M} \oplus \Delta)\mathrm{SB} \oplus IV \oplus \mathrm{c} \oplus (\mathrm{MSB} \oplus IV \oplus \mathrm{c}) = \Delta\mathrm{SB} = 0 \tag{11}$$

The matrices S and B are described in Section 3. A collision-producing difference can be found by solving the set of linear equations given in (11).

Furthermore, the following theorem shows that every near-collision [1] can be turned into a collision with only a minor increase in complexity.

Table 3. Smallest Hamming weight found for FORK-256 and reduced variants

stream	collision	near-collision
one stream	52	35
1 & 2	384	135
3 & 4	384	288
full hash	1280	704

Theorem 3. *For L-FORK-256, every two-block message difference of the form* (Δ, Δ) *produces a two-block collision.*

Proof:

$$\begin{aligned}
\text{L-FORK-256}((\mathrm{M}_1 \oplus \Delta)\|(\mathrm{M}_2 \oplus \Delta)) &= LH(LH(IV, \mathrm{M}_1 \oplus \Delta), \mathrm{M}_2 \oplus \Delta) \\
&= LH(IV \oplus (\mathrm{M}_1 \oplus \Delta)\mathrm{SB} \oplus \mathrm{c}, \mathrm{M}_2 \oplus \Delta) \\
&= (IV \oplus (\mathrm{M}_1 \oplus \Delta)\mathrm{SB} \oplus \mathrm{c}) \oplus (\mathrm{M}_2 \oplus \Delta)\mathrm{SB} \oplus \mathrm{c} \\
&= (IV \oplus \mathrm{M}_1\mathrm{SB} \oplus \mathrm{c}) \oplus \mathrm{M}_2\mathrm{SB} \oplus \mathrm{c} \\
&= \text{L-FORK-256}(\mathrm{M}_1\|\mathrm{M}_2)
\end{aligned}$$

□

4.2 Minimizing the Number of Conditions

It is difficult to bound the number of conditions that have to be fulfilled in order to guarantee that the message follows the characteristic in the original FORK-256 hash function. A commonly used approach is to use the Hamming weight of the expanded message to approximate the attack complexity. This approximation is useful for SHA-1, but does not hold in the case of FORK-256. A property of L-FORK-256 is that collision producing differences with very low weight in the message, can easily result in very high weights in the internal states of the four separate streams. Hence, to get a more accurate approximation of the final attack complexity the weight of the internal state variables has to be considered as well.

We used algorithms from coding theory to find characteristics with low Hamming weight. Even if the algorithms are probabilistic they are expected to do a good job as they did in the case of SHA-1 [4]. In Table 3, the smallest found weights for FORK-256 and reduced variants are shown.

Converting the Hamming weights to numbers of conditions is complicated by the following issues.

1. One equation may cover several conditions imposed on bits in identical positions of several registers.
2. One equation may cover conditions imposed on bits in neighboring positions of several registers.

3. Conditions imposed on bits in the MSB position of a 32-bit word may be fulfilled automatically, due to carry overflow effects.
4. Some conditions might be reworked to linear conditions involving only message bits. Such conditions are easy to fulfill and don't contribute to the probability of the characteristic.

A rough estimation of the work factor can be made by taking the Hamming weight of the internal state variables and the weight of the expanded message. For FORK-256 with all four streams, the estimate probability for a random message having the chosen differences to follow the linear characteristic and to collide is 2^{-1280}. The probability for a near-collision is 2^{-704}. These probabilities are too small to pose a realistic threat to the hash function. Note that the smallest found Hamming weight for one stream is equal to the local collision given in [2].

4.3 A Differential Characteristic for 4 Steps with Probability 1

For four (out of eight) steps of FORK-256 there exists a characteristic with probability 1. If we choose the same difference δ in all message words $m_i^* = m_i \oplus \delta$, for $i = 0, \ldots, 15$ and differences in all chaining variables $A'_{j,k} = \cdots = H'_{j,k} = \delta$ for $j = 1, \ldots, 4$ for a $k < 5$ then we have after 4 steps:

$$A'_{j,k+4} = B'_{j,k+4} = \cdots = H'_{j,k+4} = 0 \quad \text{for } j = 1, \ldots, 4$$

This characteristic holds with probability 1 for $\delta = \texttt{80000000}$. For all the other cases the probability of the characteristic is approximately $2^{-\mathrm{HW}(\delta)*12*4}$. It is difficult to use this characteristic to break FORK-256. To construct a collision we would need a characteristic (not necessary linear) for the first 4 steps in each stream that produces the needed differences in all the chaining variables.

5 Truncated Differential Attack

The function f and g map a 32-bit input word to a 32-bit output. By design, these functions are not invertible (although their linear approximations are). This means we could try to construct collisions by using values $x \neq x^*$ which have the property that $f(x) = f(x^*)$ and $g(x+\delta_k) = g(x^*+\delta_k)$. For the analysis, it is most convenient to consider as difference operation the modular difference:

$$x' = x - x^* \bmod 2^{32} \ . \tag{12}$$

5.1 One Stream

We first consider one stream of the hash function. For the attack, we want to exploit a truncated characteristic of the following form:

$$(0,0,0,0,0,0,0,0) \tag{13a}$$
$$\Downarrow \qquad x \neq x^*, f(x) = f(x^*), g(x+\delta_k) = g(x^*+\delta_k)$$
$$(0,x',0,0,0,0,0,0) \tag{13b}$$
$$\Downarrow \qquad \oplus \text{ changes the difference}$$
$$(0,0,\alpha,0,0,0,0,0) \tag{13c}$$
$$\Downarrow \qquad \oplus \text{ changes the difference}$$
$$(0,0,0,\beta,0,0,0,0) \tag{13d}$$
$$\Downarrow \qquad \oplus \text{ changes the difference}$$
$$(0,0,0,0,\gamma,0,0,0) \tag{13e}$$

The difference in the 5th register can be canceled by adding a message block with a suitable difference. Alternatively, the characteristic can be concatenated with a rotated version of itself.

The probability of the first step depends on the difference x' and on the value of the constant δ_k. There are many δ_k values for which the probability of the differential equals 0, but there is also a significant fraction for which one can find at least one x' such that the probability becomes greater than zero. We call a δ_k value *weak*, if there exists at least one difference x' for which the probability of the differential is greater than 0.

5.2 Weak Constants

By doing an exhaustive search we found 2 *weak* constants for the right side and 4 *weak* constants for the left side of one stream of FORK-256. The *weak* constants are shown in Table 4.

Table 4. Weak constants in FORK-256

side	constant	x	x^*
left side	δ_2	AEB691E5	06DEF69A
	δ_1	6FF2F3E9	4B4D2A05
	δ_3	67EAC4D8	27A61343
	δ_7	20D331A5	04549CDC
right side	δ_{10}	D73BC777	445C5563
	δ_{14}	EDFD4D5B	BE452586

These weak constants can be used to break one stream of FORK-256. To break FORK-256 with more than one stream we would need more weak constants (see Section 5.4). Therefore, we have to extend the concept of weak constants as described in the next section.

5.3 Semi-weak Constants: Extending the Idea of Weak Constants

Instead of searching for pairs x, x^* having zero differences at the output of f and g, we can extend the search to pairs x, x^* such that:

$$\begin{aligned} f(x) \oplus f(x^*) &= \Delta f \\ g(x + \delta_k) \oplus g(x^* + \delta_k) &= \Delta g \end{aligned} \tag{14}$$

and

$$\begin{aligned} \Delta f &= \Delta g \\ \Delta f \ll 5 &= \Delta g \ll 9 \\ \Delta f \ll 17 &= \Delta g \ll 21 \ , \end{aligned}$$

where the last condition is equivalent to $\Delta f = \Delta g \ll 4$. We found many pairs x, x^* and constants δ_k which fulfill (14). A subset of these are given in Table 7 and Table 8 in the appendix. We restrict the search to values Δf and Δg with low Hamming weight to keep the final attack complexity low. Note that additional conditions have to be fulfilled to guarantee that the differences at the output of f and g cancel out within one step. In detail, the probability that the differences cancel out is approximately $2^{-3 \cdot \mathrm{HW}(\Delta f)}$. Note, that for $\Delta f \neq 0$ the minimal Hamming weight is 8.

5.4 The Full Hash Function

If we consider the full hash, then we have to take into account two effects:

1. Due to the different permutations Σ_j, the message blocks enter in the different streams at different steps. This property complicates the attack.
2. Due to the final addition of the streams, we can convert near-collisions for each of the streams into collisions for the full hash. This property facilitates the attack.

If we consider a variant of FORK-256 where the output of the functions f and g are not considered, then there are no interactions between the different registers in the streams. For this variant, we can easily construct a collision. Note that for each $x_{j,i}$ which has a non-zero difference we need one weak constant to guarantee that the original FORK-256 hash function behaves like the variant. To minimize the number of (needed) *weak* constant, we have to minimize the number of differences in $x_j = (x_{j,0}, \ldots, x_{j,15})$, for $j = 1, 2, 3, 4$. In Table 5, we list the best results we found for FORK-256 and stream-reduced variants. Since we would need 12 weak constants to break the original FORK-256 hash function, this attack strategy is not applicable to FORK-256. We expect a complexity of at least $2^{12 \cdot 3 \cdot \mathrm{HW}(\Delta f)} \geq 2^{288}$ applications of the compression function to find a collision in FORK-256. However, FORK-256 reduced to 2 streams can be broken easily with this method as shown in Section 5.5.

5.5 A Collision for Two Streams of FORK-256

In this section, we present a collision for FORK-256 reduced to two streams using the attack strategy described before. In the following, we will describe how to

Table 5. Number of weak constants needed to produce a collision in FORK-256 and stream-reduced variants. Note a 'x' denotes a difference in the word $x_{j,i}$ in stream j.

differences	stream 1	stream 2	stream 3	stream 4
m_6, m_{12}	------x-----x--x	----------x-x-	-x-----xx-----x-	-x------x------x
m_2, m_{12}	--x-------xx---	-------x-----x-	-----x-x----x-x-	-x------x--x----
m_9, m_{13}	---------x---x--	---x-----xx-----	----x-x------x-x	------x---x----x
m_2, m_6	--x---x----x---x	-------x---x---	-x---x--x---x---	----------x---x
m_4, m_7	----x--x-----xx-	-------x-----xx-	x---------------	
m_0, m_{10}	x--------xx-----	-----x----x-x---		-----x---x--x---
m_3, m_8, m_9, m_{10}	---x----xx------		--x---x---------	---x----xx------
m_5, m_{10}		-----x-----xx---	--x--------xx---	x---------------
m_3, m_{12}		------x-------xx	-------x------xx	-x--------------
m_0, m_9	x---------------	---x----------		
m_{13}	------------x--		----x--------x--	
m_5	-----x------x---		----------x---	
m_6	------x--------x			--------------x
m_2	--x--------x----			----------x----
m_{12}		--------------x-	-------x------x-	
m_3		------x--------x	---------------x	
m_4		-------x------x-		--------------x-
m_9		---x------x-----		----------x-----
m_{14}, m_{15}			---x----------	----x----------

construct a collision in FORK-256 reduced to stream 1 and stream 2. Note that the attack would work similar to break FORK-256 reduced to 2 other streams.

Considering the ordering of the message words in stream 1 and stream 2 (see Table 1), we see that the distance (in number of steps) between m_0 and m_9 is 4 in both streams. Hence, we need only two weak constants in the attack. The attack can be summarized as follows.

1. Choose $x_{1,0} =$ `7AB8131D` and $x^*_{1,0} =$ `728D1B48` and calculate $B_{1,1}$ and $B^*_{1,1}$.
2. Choose $x_{2,3} =$ `E2E5A2A9` and $x^*_{2,3} =$ `A6378BEC` and calculate $F_{2,2}$ and $F^*_{2,2}$.
3. Choose suitable values for $x_{1,2}$, $x_{1,4}$ and $x_{1,6}$ such that $E'_{1,4} = -x'_{2,3}$.
4. Choose suitable values for $x_{2,5}$, $x_{2,7}$ and $x_{2,9}$ such that $A'_{2,5} = -x'_{1,0}$.
5. We have 24 conditions on $B_{1,0}, C_{1,0}, D_{1,0}$ that have to be fulfilled to guarantee that the differences at the output of f and g cancel out in step 0 of stream 1. Therefore, we use an arbitrary first message block to get suitable values for $B_{1,0}, C_{1,0}, D_{1,0}$ that satisfy all necessary conditions. To find this first message block takes at most 2^{24} hash computations.

Table 6. A colliding message pair for FORK-256 reduced to two streams

h_0	06A09E667	0BB67AE85	03C6EF372	0A54FF53A	0510E527F	09B05688C	01F83D9AB	05BE0CD19
M_0	0F427DBAA	06FBF0CB7	0413F646C	0FCE4800E	0AF327AFD	05CB1B99A	00C879908	0FD5EA595
	0BA603C95	06CF74DC6	0516E4AD5	01E43C9B5	03A112367	0258259E8	0FC3FA69D	0CD4F8D0C
h_1	06A09E667	0C1F86BBC	0D2856B94	052847CA9	0B8D977FE	0EE42EED7	0A309479B	05C5A4DA8
M_1	010AE2CB6	000000000	010ABB697	000000000	0197E717C	000000000	01FDE8BA2	000000000
	0D4A419E3	0E3082DF1	0E7C9B7DB	000000000	000000000	0B93DF199	000000000	0314E6339
M^*_1	0088334E1	000000000	010ABB697	000000000	0197E717C	000000000	01FDE8BA2	000000000
	0D4A419E3	0A65A1734	0E7C9B7DB	000000000	000000000	0B93DF199	000000000	0314E6339
M'_1	0082AF7D5	000000000	000000000	000000000	000000000	000000000	000000000	000000000
	000000000	03CAE16BD	000000000	000000000	000000000	000000000	000000000	000000000
h_2	06A09E667	06D320398	00E1A7F40	0A359E80E	0E029DE72	019F5C484	032084418	0836E2FD8
h^*_2	06A09E667	06D320398	00E1A7F40	0A359E80E	0E029DE72	019F5C484	032084418	0836E2FD8

6. We also have 24 conditions on $F_{2,1}, G_{2,1}, H_{2,1}$ in stream 2 that have to be fulfilled to guarantee that the differences at the output of f and g cancel out in step 1 of stream 2. Therefore, we have to modify $x_{2,1}$ to satisfy these conditions. We can find a suitable value for $x_{2,1}$ in at most 2^{24} trials.
7. Calculate m_i for $i = 0, \ldots, 15$ from the x-values calculated in step 1-6.

$$\begin{aligned}
m_0 &= w_{1,0} = x_{1,0} - A_{1,0} \\
m_{15} &= w_{2,1} = x_{2,1} - E_{2,0} \\
m_2 &= w_{1,2} = x_{1,2} - A_{1,1} \\
m_9 &= w_{2,3} = x_{2,3} - E_{2,1} \\
m_4 &= w_{1,4} = x_{1,4} - A_{1,2} \quad * \\
m_{10} &= w_{2,5} = x_{2,5} - E_{2,2} \\
m_6 &= w_{1,6} = x_{1,6} - A_{1,3} \\
m_4 &= w_{2,7} = x_{2,7} - E_{2,3} \quad * \\
m_{13} &= w_{2,9} = x_{2,9} - E_{2,4}
\end{aligned}$$

   Note that there are 2 conditions on m_4. To satisfy both conditions we calculate first $m_4 = w_{1,4} = x_{1,4} - A_{1,2}$ and then we use m_8 to modify $E_{2,3}$ such that $m_4 = w_{1,7} = x_{2,7} - E_{2,3}$ holds.

With this method we can easily construct collisions in the FORK-256 variant. Once we have fixed the values of the chaining variables by using an arbitrary first message block and have determined $x_{1,0}$, $x_{1,2}$, $x_{1,4}$, $x_{1,6}$, $x_{2,1}$, $x_{2,3}$, $x_{2,5}$, $x_{2,7}$, and $x_{2,9}$ we can construct many collisions by solving the system of equations given in step (7) of the attack. We can construct about $2^{(16-9)\cdot 32}$ colliding message pairs once we have fixed all the x-values. This can be compared to having 224 *neutral bits* [1] in the message. In Table 6, we give a colliding message for FORK-256 reduced to the first 2 streams.

Complexity Analysis. In this section, we will give a detailed complexity analysis of the attack on FORK-256 reduced to 2 streams. The attack basically consists of 2 parts:

1. Find the values of Table 7 and Table 8 in the appendix
2. Find suitable x-values.

The first part of the attack has complexity of at most $2^{32} \cdot 4 = 2^{34}$ computations of f and g. This is equivalent to at most 2^{28} computation of the compression function of FORK-256. Note that the real complexity might be much lower, since g is only calculated if Δf is correct. While the first part of the attack is computational expensive, the second part of the attack has a comparable low complexity. For each difference in $x_{j,i}$ for $j = 1, 2, 3, 4$ and $i = 0, \ldots, 15$ we have to fulfill 24 conditions on the chaining variables and further find 3 suitable x-values to guarantee the that the difference cancel out after 4 steps. Therefore, we need 9 x-values in the attack on the first 2 streams. To find all these x-values and calculating the first block to fulfill all conditions on the chaining variables

take us at most $7 \cdot 2^{32}$ calculations of f and g, and 2^{24} hash computations. Note that the probability for finding a suitable x-value is much higher than 2^{-32} in practice. Thus, the complexity will be much lower than 2^{29} for this part of the attack. Hence, the final attack complexity for both parts of the attack is about 2^{29} for the first collision. Many other collisions can be constructed with probability 1 afterward.

Improving/Extending the Attack. There are several ways to improve the attack. In the following we list some of them:

1. The attack can be modified to construct collisions in FORK-256 reduced to two other streams.
2. As shown by way of an example, the degrees of freedom (the number of neutral bits) we have in the attack on 2 streams is quite large. Thus, we can try to extend the attack to 3 streams of FORK-256.
3. Since the number of needed weak constants is too large for an attack on the original FORK-256 hash function, we could try to construct a near-collision in the hash function (only 6 weak constants needed).
4. Instead of searching for values (x, x^*) for which $\Delta f = \Delta g$, we can extend the search to values for which $\Delta f \neq \Delta g$, but the differences cancel out due to carries of the modular addition. Therefore, we have to find a good method to reduce the search space and the runtime for finding these values, respectively.

6 Conclusions

In this article we presented the first cryptanalytic results on the hash function FORK-256. We showed that the linearized variant of FORK-256 has several unusual properties which do not exist in the linearized variants of other hash functions. We also explained why the linearized model can not be used to mount attacks similar to the recent attacks by Wang *et al.* on SHA-like hash functions.

Furthermore, we showed how collision attacks, exploiting the non-bijectiveness of the nonlinear functions of FORK-256, can be mounted on reduced variants of FORK-256. We presented an efficient attack on FORK-256 reduced to 2 streams. Moreover, we expect that our attack can also be extended to FORK-256 reduced to 3 streams. For the moment our approach does not appear to be applicable to the full FORK-256 hash function.

However, this does not prove that FORK-256 is secure. Further analysis is required to get a good view on the security margins of FORK-256.

Acknowledgements

The authors wish to thank Christophe De Cannière, Christian Rechberger, Norbert Pramstaller, Vincent Rijmen, and the anonymous referees for useful comments and discussions.

References

1. Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.
2. Deukjo Hong, Jaechul Sung, Seokhie Hong, Sangjin Lee, and Dukjae Moon. A New Dedicated 256-bit Hash Function: FORK-256. In Matt Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Proceedings*, volume 4047 of *LNCS*, pages 195–209. Springer, 2006.
3. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.
4. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *LNCS*, pages 78–95. Springer, 2005.
5. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
6. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

A Appendix

Table 7. List of (semi) weak constants for the left side of the stream

side	constants	$\Delta f = \Delta g$	x	x^*
left side	δ_0	11111111	7AB8131D	728D1B48
	δ_0	88888888	B2D7C3DA	3B10A457
	δ_1	11111111	DC7A5519	38EC29EF
	δ_1	11111111	E02717D6	9FED7307
	δ_1	44444444	C2886A61	545D72A2
	δ_1	88888888	8DC83B78	1C547838
	δ_2	22222222	650CA295	4605419A
	δ_2	44444444	BEF57D44	2CF60FEB
	δ_3	22222222	E3E8525C	4CA6452C
	δ_3	44444444	8EA41F57	642967E8
	δ_3	44444444	7B080CA7	304EB46C
	δ_4	22222222	64B41C80	4D83EDBA
	δ_5	44444444	F92C0421	46119614
	δ_5	88888888	130F16FD	113044A8
	δ_6	22222222	8DD2989F	3FC9AB68
	δ_8	11111111	FE26B64C	CA52EA30
	δ_8	11111111	AD85682B	609F1D2F
	δ_8	22222222	BAF886F0	4BAF0F68
	δ_8	44444444	B14939BC	0D0B62B8
	δ_8	44444444	CE341C7A	AC04B7A3
	δ_9	11111111	B16A5B43	97949A93
	δ_9	44444444	FEF6F543	4D044E8C
	δ_9	88888888	7AB68B12	68C08524
	δ_{10}	88888888	F542812D	71F08875
	δ_{11}	88888888	50411ED1	23B25243
	δ_{11}	44444444	F7DF0AAC	7C65633B
	δ_{11}	22222222	4AB0742B	17E1B95C
	δ_{12}	44444444	CE44B12D	8EDD5A2B
	δ_{12}	11111111	E940C5B8	C0304FF9
	δ_{12}	44444444	86B755E0	73EF1636
	δ_{12}	11111111	5566F6BE	3F3136F2
	δ_{13}	22222222	48A7C925	279A5753
	δ_{13}	44444444	AE36E874	12D10ADA
	δ_{13}	22222222	FACB2049	F947DBD2
	δ_{14}	22222222	E545F52D	46511638
	δ_{14}	88888888	678E6534	02271592
	δ_{14}	22222222	CD454CD7	3D6A82F0
	δ_{14}	88888888	F3508338	C32F4A66
	δ_{15}	22222222	68B8B75D	46A9FF78
	δ_{15}	44444444	F7C30C12	56C94895

Table 8. List of (semi) weak constants for the right side of the stream

side	constants	$\Delta f = \Delta g$	x	x^*
	δ_0	11111111	87311631	CF174A81
	δ_0	88888888	9AE34AAD	E9BBB576
	δ_0	88888888	7078180F	CA9E34B0
	δ_1	22222222	8A7B922A	8515FD65
	δ_1	44444444	49E17C65	D2FAFF64
	δ_3	11111111	B93446E3	3AEE54AD
	δ_3	44444444	47233861	190D5338
	δ_4	22222222	5C40490B	4D886BE9
	δ_5	44444444	8673BC03	636F7E88
	δ_5	44444444	CC6F6AFE	AAF1DE10
	δ_6	22222222	249BD62F	717C851E
	δ_6	22222222	E5C43BC9	9C7E42D8
	δ_{10}	11111111	F0D362CD	E15DA3A4
	δ_{12}	11111111	E2E5A2A9	A6378BEC
right side	δ_{12}	22222222	02FCA84E	A822C4E6
	δ_{12}	22222222	F150D9B4	DA63A7EA
	δ_{12}	22222222	24193476	93C46D96
	δ_{13}	22222222	B43BA7D4	A491977E
	δ_{13}	22222222	DF3661E0	A6F79CF2
	δ_{13}	22222222	DF3661A0	A6F79CB2
	δ_{13}	11111111	28E0B213	C91908C7
	δ_{13}	44444444	13BB91E2	B7F968E6
	δ_{14}	88888888	99394D77	73F1C4C9
	δ_{14}	44444444	B5FAEFDB	6A6FE934
	δ_{14}	88888888	AC9747A5	77F40F98
	δ_{15}	11111111	1D405A4E	0BAE9B75
	δ_{15}	22222222	C0D7FE3A	53480ECC
	δ_{15}	88888888	AE4B89E3	6EDF99DA
	δ_{15}	88888888	AE0B89E3	6E9F99DA

Second Preimages for SMASH*

Mario Lamberger, Norbert Pramstaller,
Christian Rechberger, and Vincent Rijmen

Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria
{Christian.Rechberger,Vincent.Rijmen}@iaik.tugraz.at

Abstract. This article presents a rare case of a deterministic second preimage attack on a cryptographic hash function. Using the notion of controllable output differences, we show how to construct second preimages for the SMASH hash functions. If the given preimage contains at least $n + 1$ blocks, where n is the output length of the hash function in bits, then the attack is deterministic and requires only to solve a set of n linear equations. For shorter preimages, the attack is probabilistic.

Keywords: SMASH, hash functions, cryptanalysis, second preimages.

1 Introduction

So far, cryptanalysis of dedicated hash functions mainly focused on collision resistance. The goal here is to find an *arbitrary* pair of messages whose hash values collide. The cryptanalysis is usually considered to be successful if the employed method needs less than $2^{n/2}$ hash evaluations, where n is the length of the output. Recently and most notably a successful cryptanalysis of SHA-1 was shown which targets its collision resistance [12].

However, for many applications of hash functions, fast collision search attacks are not considered to be a threat. Instead, properties like pseudorandomness, one-wayness or resistance against second preimage attacks are usually expected and needed from a hash function. We know that collision resistance implies second preimage resistance [7], at least if the existence of a family of hash functions is assumed [11].

In this article we present a quite rare case of a second preimage attack on a hash function. The only known preimage or second preimage attacks in the literate are a second preimage attack on MD4 reduced to two rounds [4], a

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT, and by the Austrian Science Fund (FWF) project P18138. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 101–111, 2007.

preimage attack on MD2 [6] and a chosen-message second preimage attack on MD4 [13].

The targeted hash function of this paper was named SMASH and presented at FSE 2005 [5]. It turns out that the collision attack presented in [9] can be extended to a second preimage attack, which works for almost all hash functions designed according to the SMASH design strategy. Before that, we first review the SMASH design and list some of its properties in Section 2. The attack is based on the concept of *controllable output differences* which we introduce in Section 3. A method to turn the ability to produce controllable output differences into preimage/second preimage attacks of known hash function constructions is given there. In Section 4 we show how to construct arbitrary differences in the chaining variables of almost any instantiation of the SMASH design strategy, including SMASH-256 and SMASH-512. Subsequently, we turn this into a second preimage attack in Section 5. Finally we give some general remarks on how to avoid the identified vulnerabilities in Section 6 and conclude in Section 7.

2 The SMASH Design Method

We present here an overview of the hash function design strategy presented in [5]. Basically, we follow the notation of [5], except that we denote finite field addition by '+'.

2.1 Definition of SMASH

Knudsen [5] proposes a new hash function design strategy with a nonlinear compression function based on a bijective n-bit mapping f, where n is the output length of the hash function in bits. Let $m = m_1, m_2, \ldots, m_t$ be the message input after MD strengthening, where each block m_i consists of n bits. The hash output h_{t+1} is computed as follows where h_i denotes the chaining variable after iteration i:

$$h_0 = f(\mathrm{iv}) + \mathrm{iv} \tag{1}$$

$$h_i = f(h_{i-1} + m_i) + h_{i-1} + \theta m_i \quad \text{for } i = 1, \ldots, t \tag{2}$$

$$h_{t+1} = f(h_t) + h_t \ . \tag{3}$$

The last step (3) is equivalent to the processing of an additional message block, filled with zeroes. Hence, we can also describe this as a different padding rule, where first MD strengthening is applied and then a block m_{t+1} filled with zeroes is appended.

The multiplication by θ in (2) is defined as an operation in the finite field $\mathrm{GF}(2^n)$. Here, θ is an arbitrary field element in $\mathrm{GF}(2^n)$ with the only restriction that $\theta \notin \{0, 1\}$.

Knudsen proposes two instantiations of the new model, called SMASH-256 and SMASH-512. For instance SMASH-256 is specified by setting $n = 256$, by defining the finite field $\mathrm{GF}(2^{256})$ via the irreducible polynomial $g(\alpha)$,

$$g(\alpha) = \alpha^{256} + \alpha^{16} + \alpha^3 + \alpha + 1, \tag{4}$$

and by defining the function f. Because the attack we describe in this paper works for almost all instantiations of the SMASH model, including SMASH-256 and SMASH-512, we don't repeat the detailed description of the instantiations. In the remainder of this paper we will write SMASH to denote an instantiation of the SMASH design method.

2.2 Properties of the SMASH Structure

The structure of SMASH in (2) exhibits the following property.

Property 1 (forward prediction property [5]). Let h_{i-1}, h^*_{i-1} be two intermediate hash values. Choose a value for m_i and compute $m^*_i = m_i + h_{i-1} + h^*_{i-1}$. Then

$$h_i + h^*_i = (1 + \theta)(h_{i-1} + h^*_{i-1}) \ .$$

In our analysis of SMASH, we don't use any properties of the function f, except for the following property, which follows from the simple observation that repeated input values lead to repeated output values.

Property 2 (Deterministic function property). If $x_i = x_j$ and $x^*_i = x^*_j$ then $f(x_i) + f(x^*_i) = f(x_j) + f(x^*_j)$.

This property may sound trivial, but it has been used in [9] to construct collisions for SMASH.

3 Controllable Output Difference Implies Preimages

We say that we can control the output difference of a hash function $\mathcal{H}$ if there exists a *base message* m, and a set of *offset messages* $\mathrm{m}^*(i)$ such that for any given output difference ϵ, we can efficiently determine the offset message $\mathrm{m}^*(i_\epsilon)$ such that

$$\mathcal{H}(\mathrm{m}^*(i_\epsilon)) = \mathcal{H}(\mathrm{m}) + \epsilon. \tag{5}$$

Assume now that we want to construct a preimage for the value H. Then we proceed as follows. We compute $\epsilon = \mathrm{H} - \mathcal{H}(\mathrm{m})$. If $\epsilon = 0$, then m is a preimage for H. Otherwise, the offset message $\mathrm{m}^*(i_\epsilon)$ is a preimage, because

$$\mathcal{H}(\mathrm{m}^*(i_\epsilon)) = \mathcal{H}(\mathrm{m}) + \epsilon = \mathcal{H}(\mathrm{m}) + \mathrm{H} - \mathcal{H}(\mathrm{m}) = \mathrm{H}.$$

4 Constructing Arbitrary Differences in a Chaining Variable

In this section, we define the messages that we need in order to construct arbitrary differences in the chaining variable of SMASH.

4.1 Structured Messages

The *base message* m contains n message blocks. The first block, denoted by m_1, can be selected arbitrarily. The next blocks are defined as follows:

$$\mathrm{m}_i = \mathrm{m}_1 + \mathrm{h}_0 + \mathrm{h}_{i-1}, 1 < i \leq n, \tag{6}$$

where h_i denotes the value of the chaining variable after processing message m_i: h_0 is defined by (1) and h_i by (2).The purpose of this base message is to ensure that the inputs to the function f are the same for each iteration.

Secondly, we define the *offset messages* $\mathrm{m}^*(\delta)$, where δ is a vector of n bits denoted $\delta_1, \ldots, \delta_n$, which are not all equal to zero. We denote by h_i^* the value of the chaining variable after processing message m_i^*, again according to (1) and (2). Let x be an arbitrary n-bit block, different from zero, and let a denote the value

$$\begin{aligned} a &= f(\mathrm{m}_1 + \mathrm{h}_0) + f(\mathrm{m}_1 + x + \mathrm{h}_0) + \theta x \qquad (7)\\ &= f(\mathrm{m}_i + \mathrm{h}_{i-1}) + f(\mathrm{m}_i + x + \mathrm{h}_{i-1}) + \theta x, \qquad (8) \end{aligned}$$

where the second equality follows from the definition of the base message (6). Then the offset message $\mathrm{m}^*(\delta)$ is defined as follows:

$$\begin{aligned} \mathrm{m}_1^* &= \mathrm{m}_1 + \delta_1 x,\\ \mathrm{m}_i^* &= \mathrm{m}_i + \delta_i x + a \textstyle\sum_{j=1}^{i-1} \delta_j (1+\theta)^{i-j-1},\ 1 < i \leq n. \end{aligned} \tag{9}$$

4.2 Constructing the Desired Offset Message

Using Property 1 and Property 2, the following theorem can be shown, which is also given without proof in [9].

Theorem 1. *For any non-zero value x, defining m, δ, a and $\mathrm{m}^*(\delta)$ as above, it holds that:*

$$\mathrm{h}_n + \mathrm{h}_n^* = a \sum_{j=1}^{n} \delta_j (1+\theta)^{n-j}.$$

Proof. We give a proof by induction, showing that:

$$\mathrm{h}_t + \mathrm{h}_t^* = a \sum_{j=1}^{t} \delta_j (1+\theta)^{t-j},\ 1 \leq t \leq n. \tag{10}$$

In the first step, we set $t = 1$. If $\delta_1 = 0$, then we have from (9) that $\mathrm{m}_1^* = \mathrm{m}_1$ and hence $\mathrm{h}_1 + \mathrm{h}_1^* = 0$. If $\delta_1 = 1$, then $\mathrm{m}_1^* = \mathrm{m}_1 + x$ and

$$\begin{aligned} \mathrm{h}_1 + \mathrm{h}_1^* &= (f(h_0 + \mathrm{m}_1) + h_0 + \theta \mathrm{m}_1) + (f(h_0 + \mathrm{m}_1 + x) + h_0 + \theta(\mathrm{m}_1 + x))\\ &= a. \end{aligned}$$

Assume now that (10) holds for index $t \geq 1$, then we need to show it also holds for index $t+1$. Applying (9) and the induction hypothesis gives:

$$\mathrm{m}_{t+1} + \mathrm{m}^*_{t+1} + \delta_{t+1}x = a\sum_{j=1}^{t} \delta_j(1+\theta)^{t-j} = \mathrm{h}_t + \mathrm{h}^*_t. \tag{11}$$

Hence, if $\delta_{t+1} = 0$, then we have:

$$\begin{aligned}
\mathrm{h}_{t+1} + \mathrm{h}^*_{t+1} &= \underbrace{f(\mathrm{h}_t + \mathrm{m}_{t+1}) + f(\mathrm{h}^*_t + \mathrm{m}^*_{t+1})}_{0} + \mathrm{h}_t + \mathrm{h}^*_t + \theta(\mathrm{m}_{t+1} + \mathrm{m}^*_{t+1}) \\
&= (1+\theta)a\sum_{j=1}^{t} \delta_j(1+\theta)^{t-j} \\
&= a\sum_{j=1}^{t+1} \delta_j(1+\theta)^{t+1-j}.
\end{aligned}$$

If $\delta_{t+1} = 1$, then we use (8) and (11) to get:

$$\begin{aligned}
\mathrm{h}_{t+1} + \mathrm{h}^*_{t+1} &= f(\mathrm{h}_t + \mathrm{m}_{t+1}) + f(\mathrm{h}^*_t + \mathrm{m}^*_{t+1}) + \mathrm{h}_t + \mathrm{h}^*_t + \theta(\mathrm{m}_{t+1} + \mathrm{m}^*_{t+1}) \\
&= (a + \theta x) + a\sum_{j=1}^{t} \delta_j(1+\theta)^{t-j} + \theta\left(a\sum_{j=1}^{t} \delta_j(1+\theta)^{t-j} + x\right) \\
&= a\left(1 + (1+\theta)\sum_{j=1}^{t} \delta_j(1+\theta)^{t-j}\right) = a\sum_{j=1}^{t+1} \delta_j(1+\theta)^{t+1-j}
\end{aligned}$$

□

The proof is based on the fact that (6) and (9) have the following effect on the inputs to f.

Let $y_i = m_i + h_{i-1}$ be the inputs to f when processing the base message m. Similarly, let y^*_i be the inputs to the compression function when processing the offset message m^*. In case of the base message m, we have

$$\forall i, 1 \leq i < n : y_i = m_1 + h_0 \ .$$

In case of the offset message m^*, the inputs to f depend on δ, but still can only have two values:

$$\forall i, 1 \leq i < n : y_i = m_1 + h_0 + \delta_i \cdot x \ .$$

Theorem 1 can be used to compute the value $\mathrm{h}_n + \mathrm{h}^*_n$ corresponding to a given base message m, difference x and vector δ. Conversely, when given a value for m, x and $\mathrm{h}_n + \mathrm{h}^*_n$, the corresponding value for δ can be computed by solving a set of n linear equations in the unknowns $\delta_1, \ldots, \delta_n$. If the polynomials $(1+\theta)^i \bmod g(\alpha)$, $0 \leq i < n$, are linearly independent, then there is always exactly one solution. Once δ is known, the corresponding offset message m^* can be constructed using (9).

The polynomials $(1+\theta)^i$ are independent if the element $(1+\theta)$ is one of the elements that are not in a proper subfield of $\mathrm{GF}(2^n)$. If n is a power of 2, then the number of such elements is $2^n - 2^{n/2}$. Hence, a randomly selected θ will produce independent polynomials with overwhelming probability. This is the only condition that is required for our attack to work deterministically on a hash function designed according to the SMASH design strategy.

4.3 SMASH-256 and SMASH-512

In the cases of SMASH-256 and SMASH-512 we have that $\theta = \alpha$, and $(1+\alpha)$ is not in a proper subfield. For these cases, the value δ can be computed with Algorithm 1, which is faster than the general method to solve linear equations.

Algorithm 1. Compute δ for SMASH

Require: First preimage M
Ensure: $\delta_i \in \{0, 1\}$
  Compute base message m as described in (6)
  Compute a as described in (7)
  $V \Leftarrow (\mathrm{h}_{n+1} + \mathrm{h}^*_{n+1})a^{-1}$
  Compute the polynomial representation $V(\alpha)$ of V
  Let $T_n(\alpha) \Leftarrow V(\alpha)$
  $i \Leftarrow n$
  Initialize δ with 0
  repeat
    Perform the polynomial division $T_i(\alpha) = T_{i-1}(\alpha)(1+\alpha) + t_i$ to determine the quotient $T_{i-1}(\alpha)$ and the remainder t_i.
    $\delta_i \Leftarrow t_i$
    $i \Leftarrow i - 1$
  until $i = 0$ or $T_i(\alpha) = 0$
  {Check that $t_i \in \{0, 1\}, \forall i$, that $\deg(T_i(\alpha)) \leq i$, $\forall i$ and that $T_i = 0$.}

In [9], collisions for SMASH were constructed by choosing an arbitrary base message and selecting a proper offset message. Observe that for n-block messages, $\mathrm{h}_n + \mathrm{h}^*_n = 0$ always leads to the unacceptable solution $\delta = 0$. That is the reason why $n+1$-block messages (without considering the padding) have to be used in order to construct collisions.

5 Construction of Second Preimages

Consider first a reduced variant of SMASH, which is defined by leaving out the padding (we only consider inputs with a length that is a multiple of n) and by leaving out the final application of the compression function (3). The discussion in the previous section implies that we can control the output difference of this SMASH variant. Hence, we can construct preimages. In order to do this, we need to control the message blocks $\mathrm{m}^*_2, \ldots, \mathrm{m}^*_n$. This is illustrated in Fig. 1.

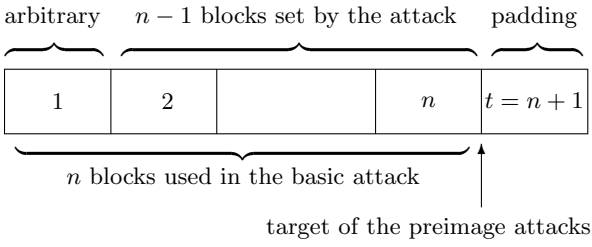

Fig. 1. Structure of the messages used in the (second) preimage attacks

Due to the padding and the final processing in SMASH, we can't control the message inputs for the last two applications of the compression function. We see currently no way to find preimages for SMASH. However, we can construct second preimages.

For simplicity, assume we are given a message M (after padding) with length $t = n+1$ blocks, *i. e.* the first preimage. Later on we will show how to extend the approach to longer and also shorter messages. We can easily compute H_{t-1}, the value of the chaining variable before the processing of block M_t. Now we construct an n-block preimage for H_{t-1}. Subsequently we concatenate M_t to produce a second preimage for SMASH(M). This attack produces second preimages of the same length as the given preimage.

5.1 Second Preimages for Longer Messages

For messages of length $t \geq n+2$, there are two possibilities to construct second preimages using the described attack.

1. The target for the preimage is again H_{t-1}. Choose arbitrary values in the first $t-n$ blocks instead of the first block only and then do the attack by controlling the last $n-1$ message blocks. The resulting message structure is illustrated at the top of Fig. 2.
2. Instead of constructing a preimage for H_{t-1}, construct a preimage for H_n instead. Only the first block can be chosen arbitrarily, the last $t-n$ blocks are the same for the first and the second preimage. The resulting message structure is illustrated at the bottom of Fig. 2.

5.2 Second Preimages for Shorter Messages

The attack can be extended to find also second preimages for a first preimage M with length $t \leq n$. Following Theorem 1, we then get n equations in $t-1$ unknowns:

$$\mathrm{h}_{t-1} + \mathrm{h}^*_{t-1} = a \sum_{j=1}^{t-1} \delta_j (1+\theta)^{t-1-j}. \tag{12}$$

With probability 2^{t-1-n}, there exists a solution δ. If no solution exists, then the attack has to be repeated with another value for m or x. On average, the attack

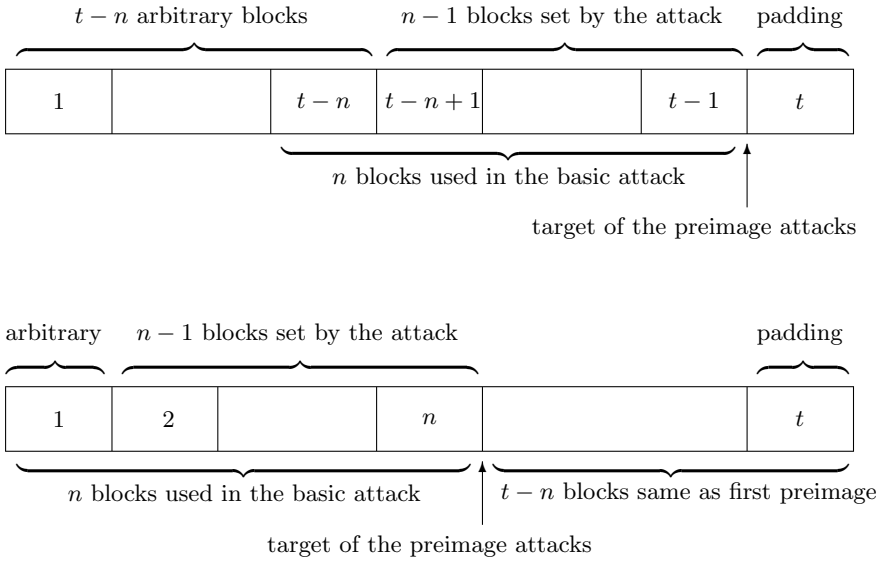

Fig. 2. Two different structures of the messages used in the (second) preimage attacks for longer messages

will succeed after 2^{n+1-t} iterations. Hence the presented attack is faster than the general meet-in-the-middle attack [5] for messages longer than $n/2+1$ blocks.

5.3 Summary of Results

Table 1 summarizes the results on second preimage attacks on SMASH.

Table 1. Overview of second preimage attacks on SMASH

type	message length t	number of blocks the attacker can choose	probability
meet-in-the-middle [5]	≥ 2	$t-2$	$2^{-n/2}$
this paper	$\geq n+1$	$t-n$	1
this paper	$< n+1$	1	2^{t-1-n}

6 Additional Discussions

Subsequently we discuss general ways to prevent the found weaknesses in SMASH in new hash function designs. In order to have a secure iterated hash function, its compression function and the way it is used should have (at least) the following properties:

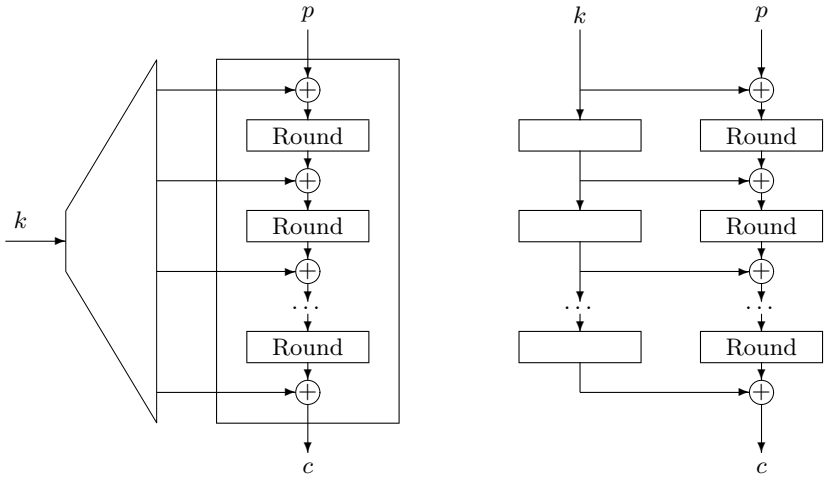

Fig. 3. Two views on a block cipher

1. there should be no 'simple' relation between input (differences) and output (differences),
2. it shouldn't be possible to apply the same input twice to the same function.

The first property is captured in the notion 'random oracle'. The second property is essential in a universal hash function.

There are currently no indications that the SMASH-256 and SMASH-512 constructions violate the first property. Note that for algorithms like SHA-1 [8] and its predecessors these relations have been found. They have been used to construct collisions [12] but they can not be used to construct controllable differences as needed for the presented second preimage attack.

However, the second property is absent from the SMASH design structure. There seem to be two alternative approaches to achieve the second property. The first one is used in block-cipher based compression functions as studied in [2,10]. A block cipher has two inputs, and the secure constructions all ensure it is impossible to control both inputs. An attacker can for instance control the plaintext input, but then the key input doesn't stay constant over the different iterations, which implies that the attacker sees different functions of the family.

A second approach is to base the compression function on a primitive that takes in principle inputs of larger size than a message block. For instance, in Alpha-MAC [3], only 32 bits of the 128-bit AES round transformation are determined by the message.

Both approaches are in fact not that very different from one another. The two inputs of a block cipher can also be seen as two parts of one input, of which one part is controllable and the other part not, see Fig. 3. In the case of 128-bit AES, the round transformation and the key schedule can be composed into a new 256-bit round transformation, that transforms both the message and the key input. This new round transformation is however highly asymmetric. Much

more symmetry is in the enlarged round transformation of the block cipher W, used in the hash function Whirlpool [1]. This is because the key schedule of W uses the round transformation. A remaining asymmetry is however the fact that while there is information flow from the 'key part' to the 'message part', no information flows in the opposite direction (see right hand side of Fig. 3).

7 Conclusion

Using the concept of controllable output differences, second preimages for SMASH can be constructed. The conjectured security level against second preimage attacks was $2^{n/2}$ hash operations. Our attack breaks this bound for messages longer than $n/2 + 1$ blocks.

We discussed how to avoid these vulnerabilities in new hash function constructions. It would be interesting to see whether there are other hash function constructions that are vulnerable to this attack.

Acknowledgements

We would like to thank Lars Knudsen and the anonymous reviewers for helpful comments.

References

1. Paulo S.L.M. Baretto and Vincent Rijmen. The Whirlpool Hashing Function. Cryptology ePrint Archive, Report 2005/281, 2000. revised in May 2003, `http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html`.
2. John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *LNCS*, pages 320–335. Springer, 2002.
3. Joan Daemen and Vincent Rijmen. A New MAC Construction ALRED and a Specific Instance ALPHA-MAC. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Proceedings*, volume 3557 of *LNCS*, pages 1–17. Springer, 2005.
4. Hans Dobbertin. The First Two Rounds of MD4 are Not One-Way. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE'98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *LNCS*, pages 284–292. Springer, 1998.
5. Lars R. Knudsen. SMASH - A Cryptographic Hash Function. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Proceedings*, volume 3557 of *LNCS*, pages 228–242. Springer, 2005.

6. Lars R. Knudsen and John Erik Mathiassen. Preimage and Collision Attacks on MD2. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Proceedings*, volume 3557 of *LNCS*, pages 255–267. Springer, 2005.
7. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. Available online at `http://www.cacr.math.uwaterloo.ca/hac/`.
8. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.
9. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Breaking a New Hash Function Design Strategy Called SMASH. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *LNCS*, pages 233–244. Springer, 2006.
10. Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *LNCS*, pages 368–378. Springer, 1994.
11. Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *LNCS*, pages 371–388. Springer, 2004.
12. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
13. Hongbo Yu and Gaoli Wang and Guoyan Zhang and Xiaoyun Wang. The Second-Preimage Attack on MD4. In Yvo Desmedt and Huaxiong Wang and Yi Mu and Yongqing Li, editors, *Cryptology and Network Security, 4th International Conference, CANS 2005, Xiamen, China, December 14-16, 2005, Proceedings*, volume 3810 of *LNCS*, pages 1–12. Springer, 2005.

A Practical Optimal Padding for Signature Schemes

Haifeng Qian[1], Zhibin Li[1], Zhijie Chen[2], and Siman Yang[2]

[1] Computer Science & Technology Department, East China Normal University, Zhongshan Road 3663 (N), Shanghai 200062, China
{hfqian,lizb}@cs.ecnu.edu.cn
[2] Department of Mathematics, East China Normal University, Zhongshan Road 3663 (N), Shanghai 200062, China,
{zjchen,smyang}@math.ecnu.edu.cn

Abstract. A digital signature scheme that achieves an optimal bandwidth (generating signatures as short as possible) is called an optimal signature scheme. The previous optimal signature schemes all need the random permutations (or the ideal ciphers) with large block size as building blocks. However, the practical cipher with large block size such as Halevi and Rogaway's CMC-mode should call the underlying secure block cipher with small block size many times each time. This makes the previous optimal signature schemes which use the large domain permutation (or the ideal cipher) less efficient in the real world, even if there exist the methods that can encipher the messages with larger domain. On the other hand, all the practical signature schemes are not optimal in bandwidth including PSS-R, FDH, DSA, etc. Hence, the problem on how to design a practical, efficient and optimal signature scheme remains open.

This paper uses two random oracles and an ideal cipher with a smaller block size to design an optimal padding for signature schemes. The ideal cipher in our scheme can be implemented with a truly real block cipher (e.g. AES). Therefore, we provide a perfect solution to the open problem. More precisely, we design a practical, efficient and optimal signature scheme. Particularly, in the case of RSA, the padding leads the signature scheme to achieve not only optimality in bandwidth but also a tight security.

Keywords: Optimal Signature, Tight Security, Random Oracle Model, Ideal Cipher Model, Short Signature.

1 Introduction

How to constructing short signatures is an important old problem since short signatures are very useful in practice. The size of the signature is also one of the measures of the efficiency of a digital signature scheme. Therefore, several proposals are shown on how to shorten the signatures while preserving a high level of security [5,6]. Another technique proposed for reducing the signature length is to design signatures with message recovery [3,21,13]. In such systems

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 112–128, 2007.

one encodes a part of the message into the signature thus shortening the total length of the message/signature pair.

Generally speaking, a digital signature scheme is a tuple of probabilistic algorithms which enable the signer to transform an arbitrary message into a signed message, such that anyone can check the validity of the signed message[13]. A signed message contains the message, plus some information to prove its validity. If in a scheme without message recovery, the signed message is the concatenation of the message and of a signature. The message expansion of a signature scheme is the difference between the length of the signed message and the original message. It is the length of the signature, if there is no message recovery.

The security level of a properly designed signature scheme is determined by not only the hardness of the underlying mathematical problem but also the size of message expansion of the signature scheme. For a specific signature scheme, one obvious lower bound of the security level is that k-bit signatures (or more exactly k bits of message expansion) cannot provide better than k bits of security, since the probability that a signature is valid is at least 2^{-k}, assuming that the underlying hard problem is $k' \geq k$ bits secure [13]. Hence, the problem on how to obtain message expansion as small as possible while still preserving high level of security has been put forward in [13]. Now it attracts more and more cryptologists' attention [17,4].

1.1 Optimal Paddings for Signature Schemes with Message Recovery

To solve the problem mentioned above, many researchers are looking for the signature schemes which have with optimal message recovery length. Granboulan [13] has shown a theorem which says that if we want a k-bit secure signature scheme, the message expansion should be at least k bits, and the signature scheme must have unique signature for each message. The author also proposed a signature scheme in the random permutation model named Basic OPSSR and improved version in the ideal cipher model, named (OPSSR), respectively. Both of the schemes achieve the lower bound of the message expansion.

However, the signature schemes' security is not tightly related to the hardness of the underlying cryptographic assumption since the underlying trapdoor permutation is required to be $2k$-bit secure if each signature scheme has k bits of message expansion and security [1]. Hence, the signature schemes are both with loose security. This makes the signature schemes both inefficient according to the theoretically secure parameters.

Katz and Wang extended Granboulan's result and showed a signature with optimal message recovery length, based on the claw-free permutations in the random permutation model [17]. Different from Granboulan's signature schemes, their signature scheme's security is tightly related to the hardness of breaking a pair of claw free permutation. The scheme is essentially optimal in terms of

[1] Even using a random-self-reducible permutation, the underlying trapdoor permutation is still required to be $(k + q_s)$-bit secure.

the allowable message length, but not with unique signature for each message. The signature can achieve the highest security level due to the tight security reduction.

More recently, Chevallier-Mames, Phan and Pointcheval proposed a universal padding, based on the random permutation in [4], named OPbP. The padding is shown to be optimal both for signing and encrypting. When the padding is used to sign message, it is optimal in bandwidth of message recovery, similar to Katz and Wang's construction.

1.2 Idealized Models and Practical Security

To our knowledge, many of signature schemes are proven secure in the following three idealized models: the random permutation model, ideal cipher model and random oracle model. An idealized (oracle) model replaces some components of the algorithm with the oracles simulated by the reduction. The number of calls to the oracles is often assumed to be upper bounded since the actual computation of the idealized components takes time. For a signature scheme the number of all the queries to the oracle is always upper bounded as well.

In fact, security proofs of the signature schemes are the descriptions on how to construct a reduction algorithm which reduces from forgery to the underlying hard problem. A reduction algorithm in an idealized model always gives random answers taken from the set of values that are consistent with previous answers. Although proofs in such models do not guarantee security when the oracles are instantiated by any particular cryptographic primitive [7], it is widely believed that a proof in an idealized model gives some confidence in the design of a cryptographic primitive [18].

1.3 Motivation

As far as we know, all the optimal signature schemes need the random permutations (or the ideal ciphers) with a block size same as the size of element that the underlying trapdoor way permutations act on. According to [13,17], we are not aware of any appropriate way to instantiate the random permutation (or the ideal cihper) for large block sizes even for 1024-bit modulus RSA [22] by now. Even more, Granboulan's OPSSR, based on an ideal cipher, is suggested to use at least 1536-bit modulus for the case of RSA. Later, the practical ciphers with large block size such as Halevi and Rogaway's CMC-mode were proposed [14]. However, such an encipher scheme in [14] employs a secure underlying block cipher with small block size and enciphering a plain-text once must call the underlying cipher many times. This makes the previous optimal signature schemes which use large domain permutation (or the ideal cipher) less efficient in the real world even if there exist the methods that can encipher the messages with larger domain.

So the previous schemes are not so efficient and practical in the real world. The problems occur with Katz and Wang's optimal signature scheme in [17] and Chevallier-Mames et al.'s universal padding based on the random permutation

(Optimal Permutation-based Padding OPbP) when the padding is used to sign on messages.

Meanwhile, all the practical signature schemes are not optimal in bandwidth including PSS-R, DSA and some other signature schemes with message recovery used as the standards now. Most of the signature schemes in the random oracle [2] are not optimal in bandwidth. The universal padding, based on the OAEP-3 rounds recently proposed by Chevallier-Mames, Phan and Pointcheval is not optimal when it is used to sign on messages as well. Hence, how to design a more practical, efficient and optimal signature scheme remains open.

1.4 Our Contribution

This paper uses two random oracles and an ideal cipher with a smaller block size to design the optimal padding for signature schemes. In the real world the ideal cipher we use can be implemented with a truly real block cipher and the random oracles can be replaced by the cryptographic hash functions. Therefore, we provide a perfect solution to the open problem. That is to say, we design a practical, efficient and optimal padding for signature schemes. We show that the padding is provably secure if the underlying trapdoor permutation is secure. The padding for signature schemes leads the signature schemes to achieve tight security reductions when the underlying trapdoor permutations are induced by a claw free permutations. Particularly, in the case of RSA, the padding makes the signature scheme achieve not only optimality in bandwidth but also a tight security reduction. We believe that it will be preferred to RSA-PSSR in the future.

2 Basic Notions of Signature Schemes

This section reviews the definition of signature scheme, the notion of security for the digital signatures and the basic cryptographic assumptions.

2.1 Definitions of Signature Schemes

We review the functional definitions of both general signature schemes as well as those supporting message recovery and the definition of security.

Definition 1. *A signature scheme is a tuple of probabilistic algorithms (Gen, Sign, Ver) over a message space $\mathcal{M}$ such that:*

- *On input 1^k where k is the security parameter, the key-generation algorithm Gen outputs a public key pk and a secret key sk.*
- *The signing algorithm Sign takes as input secret key sk and message $m \in \mathcal{M}$ and returns a signature σ.*
- *If message recovery is not supported, the verification algorithm Ver takes as input a public key pk, a message $m \in \mathcal{M}$, and a signature σ and returns accept or reject.*

– *If message recovery is supported, verification algorithm* Ver *takes as input public key* pk *and signature* σ *and returns either a message* $m \in \mathcal{M}$ *or reject.*

We make the standard correctness requirement, given here for schemes supporting message recovery (the other case is analogous): for all (sk, pk) *output by* Gen *and all* $m \in \mathcal{M}$ *we have* $\mathsf{Ver}_{pk}(\mathsf{Sign}_{sk}(m)) = m$.

Existential Unforgeability under adaptive Chosen-Message Attacks (EUF-CMA) [12] is a widely accepted standard notion for the security of digital signatures. In the following, we use a generalized version of EUF-CMA, named Strong Existential Unforgeability under an adaptive Chosen-Message Attack (SEUF-CMA) [1,17,19].

Definition 2 (SEUF-CMA). *A forger* $\mathcal{F}$ $(t, q_s, q_h, \varepsilon)$*-breaks a signature scheme if* $\mathcal{F}$ *runs in time* t*, makes at most* q_s *signature queries and at most* q_h *hash queries; and furthermore*

$$\mathbf{Pr}\left[(pk, sk) \leftarrow \mathsf{Gen}(1^k); (m, \sigma) \leftarrow \mathcal{F}^{\mathsf{Sign}_{sk}(\cdot)}(pk) : \sigma \notin \Sigma^* \wedge \mathsf{Ver}_{pk}(m, \sigma) = 1\right] \geq \varepsilon$$

where Σ^* *is the set of signatures received from the signing oracle.*

If the signature scheme supports message recovery, the definition is as above except that we are interested in the probability that $\mathcal{F}$ *outputs a signature* σ *such that* $\mathsf{Ver}_{pk}(\sigma) = m$ *but* σ *was never the response of a query to the signing oracle. That is*

$$\mathbf{Pr}\left[(pk, sk) \leftarrow \mathsf{Gen}(1^k); \sigma \leftarrow \mathcal{F}^{\mathsf{Sign}_{sk}(\cdot)}(pk) : \sigma \notin \Sigma^* \wedge \mathsf{Ver}_{pk}(\sigma) = m\right] \geq \varepsilon$$

In either case, a signature scheme is $(t, q_s, q_h, \varepsilon)$*-secure if no forger can* $(t, q_s, q_h, \varepsilon)$*-break it.*

2.2 Underlying Complexity Assumptions

Trapdoor (One-Way) permutations are among the most popular cryptographic tools to design encryption schemes, signature schemes and protocols. The explicit complexity definition of trapdoor permutations is defined as follows.

Definition 3 (Trapdoor Permutations). *A trapdoor permutation family is a tuple of probabilistic polynomial time (*PPT*) algorithms (*Gen*,* Eval*,* Invert*) such that:*

1. $\mathsf{Tp\text{-}Gen}(1^k)$ *outputs a pair* (f, f^{-1})*, where* f *is a permutation over* $\{0,1\}^k$.
2. $\mathsf{Eval}(1^k, f, x)$ *is a deterministic algorithm which outputs some* $y \in \{0,1\}^k$ *(assuming* f *was output by* Gen *and* $x \in \{0,1\}^k$*). We will often simply write* $f(x)$ *instead of* $\mathsf{Eval}(1^k, f, x)$.
3. $\mathsf{Invert}(1^k, f^{-1}, y)$ *is a deterministic algorithm which outputs some* $x \in \{0,1\}^k$ *(assuming* f^{-1} *was output by* Gen *and* $y \in \{0,1\}^k$*). We will often simply write* $f^{-1}(y)$ *instead of* $\mathsf{Invert}(1^k, f^{-1}, y)$.

4. *(***Correctness.***) For all k, all (f, f^{-1}) output by* Gen*, and all $x \in \{0,1\}^k$ we have $f^{-1}(f(x)) = x$.*

A PPT *algorithm $\mathcal{A}$ is said to (t, ε)-break a family of trapdoor permutations if $\mathcal{A}$ runs in time at most t and outputs the preimage of a random chosen $y \in \{0,1\}^k$ for f generated by $\mathsf{Gen}(1^k)$ with probability greater than ε:*

$$\mathbf{Pr}\left[(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k); y \in_R \{0,1\}^k; x \leftarrow \mathcal{A}(1^k, f, y) : f(x) = y\right] \geq \varepsilon.$$

A family of trapdoor permutations is (t, ε)-secure if no algorithm can (t, ε)-break it.

The existence of claw-free permutations seems be reasonable. In fact, any random self-reducible permutation can be seen as a trapdoor permutation induced by a claw-free permutation [10] and almost all known examples of trapdoor permutations are self-reducible. Meanwhile, the trapdoor permutations induced by claw-free permutations can be used to obtain tight reductions[17].

Definition 4 (Claw-Free Permutations). *A family of claw-free permutations is a tuple of algorithms $\{\mathsf{Gen}; f_i; g_i | i \in I\}$ for an index set I such that:*

1. Gen *outputs a random index i and a trapdoor td.*
2. *f_i, g_i are both permutations over the same domain D_i.*
3. *there is an efficient sampling algorithm which, on index i, outputs a random $x \in D_i$.*
4. *f_i^{-1} (the inverse of f_i) and g_i^{-1} (the inverse of g_i) are both efficiently computable given the trapdoor td.*

A claw is a pair (x_0, x_1) such that $f(x_0) = g(x_1)$. Probabilistic algorithm $\mathcal{A}$ is said to (t, ε)-break a family of claw-free permutations if $\mathcal{A}$ runs in time at most t and outputs a claw with probability greater than ε:

$$\mathbf{Pr}\left[(i, td) \leftarrow \mathsf{Gen}(1^k); (x_0, x_1) \leftarrow \mathcal{A}(1^k) : f_i(x_0) = g_i(x_1)\right] \geq \varepsilon.$$

A family of claw-free permutations is (t, ε)-secure if no algorithm can (t, ε)-break it.

We say a signature scheme is k-bit secure when there is no forger can (t, q_s, ε)-break the scheme with $\log_2(t/\varepsilon) \leq k$. In the same manner, we say a trapdoor permutation (including the claw-free permutation) or a mathematic problem is k-bit secure if there is no adversary can (t', ε')-break it with $\log_2(t'/\varepsilon') \leq k'$. The value k (resp., k') also depends of the time unit used for t (resp., t'). A reduction is said to be tight if $t/\varepsilon \approx t'/\varepsilon'$. Hence, a tight reduction from forgery to the hard problem implies that the scheme achieves the same security level as the underlying hard (mathematic) problem.

3 Efficient Optimal Paddings for Signature Schemes

Various paddings for signature schemes (with message recovery) were proposed through the last few years [3,4,8,9,11,13,17,19,20]. Most of them are proven secure in the random oracle model. However, none of the schemes proven secure

in the random oracle model achieves optimality in bandwidth including PSS-R, Katz and Wang's improved PSS-R in [17]. As far as we know, all the optimal signature schemes with message recovery are based on the random permutation or the ideal cipher for large block sizes (i.e., block sizes larger than the block size of a cipher such as AES)[13,17]. Thus, the optimal signature paddings become less efficient due to the cause that larger domain cipher calls the underlying block cipher many times[14].

In the following, we propose a more practical, efficient and optimal padding for signature schemes with message recovery. This padding uses a block cipher with smaller size which can be implemented with the true block ciphers. Next, we shall prove the security of the padding for signature schemes in the idealized models: the ideal cipher model and the random oracle model.

3.1 Our Proposal

The padding for signature schemes uses a keyed permutation $E(\cdot) : \{0,1\}^l \times \{0,1\}^m \to \{0,1\}^m$, that we assume to behave like a truly ideal cipher. For each key $\kappa \in \{0,1\}^l$ of the block cipher E defines a random permutation $E_\kappa = E(\kappa, \cdot)$ on $\{0,1\}^m$. The ideal cipher E accepts both forward queries (E) as well as inverse queries (E^{-1}). Let $\varphi : \{0,1\}^n \to \{0,1\}^n$ be a trapdoor one-way permutation (or a claw-free permutation) whose inverse is φ^{-1}. The scheme uses two random oracles $H : \{0,1\}^* \to \{0,1\}^l$ and $G : \{0,1\}^m \to \{0,1\}^{n-m}$, where we assume $n > m > k$. Finally, in the following $\mathsf{PRF}_\theta(\cdot)$ designs a pseudorandom function that uses a secret key θ and generates one bit. The symbol " $\|$ " denotes the bit-string concatenation. The signature scheme is described as follows:

Key Generation Algorithm Gen: runs $\mathsf{Tp\text{-}Gen}(1^n)$ to obtain a pair of trapdoor permutation φ and φ^{-1}. Sets (φ, φ^{-1}) as the public key and secret key, respectively.

Signature Algorithm Sign: The space of the messages is $\mathcal{M} = \{0,1\}^{n-m} \times \{0,1\}^{m-k}$, the signature algorithm outputs a signature σ into $\{0,1\}^n$: on a message $M = (m_1, m_2) \in \mathcal{M}$, one computes $b = \mathsf{PRF}_\theta(M)$, $h = H(m_1)$, $\omega = E_h(m_2 \,\|\, b^k)$, $s = G(\omega) \oplus m_1$ and then $\sigma = \varphi^{-1}(s \,\|\, \omega)$.

Verification AlgorithmVer: On a signature σ, one first computes $s \,\|\, \omega = \varphi(\sigma)$ where $s \in \{0,1\}^{n-m}$, $\omega \in \{0,1\}^m$ and then $m_1 = G(\omega) \oplus s$, $h = H(m_1)$, $(m_2 \,\|\, v) = E_h^{-1}(\omega) \in \{0,1\}^{m-k} \times \{0,1\}^k$. Finally, if $v = 0^k$ or $v = 1^k$, the verifier returns the message $M = (m_1, m_2)$; otherwise returns reject.

Our scheme uses an ideal cipher with small block size which can be implemented by a real block cipher. Due to this point, the signature scheme works effectively and efficiently in the real world. All the previous optimal signature schemes use a full domain permutation. However, the existence of such a large block cipher is still an open problem according to [13,17]. Therefore, the previous optimal signature schemes are theoretically effective and can't be implemented directly by any existing block ciphers.

The following subsection presents the security proof of the signature scheme. We observe that the signature schemes with this padding can be proven secure

in the random oracle model and the ideal cipher model. The security of the signature is tightly related to the security of the underlying trapdoor permutation when the trapdoor permutation is induced by a claw-free permutation.

3.2 Security Analysis

In the following, we prove that the security of the proposed scheme is related to the security of the underlying trapdoor one-way permutation in the ideal cipher model and the random oracle model.

Theorem 1. *Let $\mathcal{F}$ be adaptively chosen-message (to the signing oracle) adversaries, against the signature scheme. Let us assume that $\mathcal{F}$ can produce an existential forgery according to the definition 2, with success probability ε (within a time bound t, after $q_e = q_E + q_{E^{-1}}$, q_s, q_h and q_g queries to the keyed permutation oracles (which includes q_E forward queries and $q_{E^{-1}}$ inverse queries), signing oracle, H-oracle and G-oracle, respectively). Then the underlying trapdoor permutation φ can be inverted with probability ε' within time $t' \leq t + (q_E + q_s + 1)T_\varphi$ where:*

$$\varepsilon' \geq \frac{1}{q_E + q_s + 1}\left(\varepsilon - \frac{q_{E^{-1}}}{2^{k-1}} - \frac{3(q_h + q_s + 1)^2}{2^{l+1}} - \frac{9(q_g + q_h + q_s + 1)\cdot(q_E + q_g + q_s + 1)}{2^{n-m+1}}\right). \quad (1)$$

Proof. We prove the theorem, with incremental games, to reduce the inversion of the permutation φ on a random instance y (i.e., find μ such that $y = \varphi(\mu)$) to an attack against the signature scheme. We show that the forger (or adversary) algorithm $\mathcal{F}$ can help us to invert $\varphi(x)$.

- Game $\mathbf{G}_0$: This is the attack game, in the ideal cipher model and the random oracle model. Several oracles are available to the adversary during the game: the ideal cipher oracles (both (E) and (E^{-1})), two random oracles ($H(\cdot)$ and $G(\cdot)$) and the signing oracle $\mathsf{Sign}_{\varphi^{-1}}(\cdot)$.

  To break the signature, the adversary $\mathcal{F}$ outputs its forgery, one checks whether it is actually valid or not. We denote by Forge_0 the event this forged signature is valid and use the same notation Forge_j in any game $\mathbf{G}_j$.

  Note that if the adversary asks q_s signature queries to the signing oracle $\mathsf{Sign}_{\varphi^{-1}}(\cdot)$, and $q_e = q_E + q_{E^{-1}}$ queries to the ideal cipher oracles, at most $q_s + q_E + 1$ queries for encryption are asked to the ideal cipher during this game, since each signing query may cause a new queries to the cipher oracle. Hence, $\varepsilon = \mathbf{Pr}[\mathsf{Forge}_0]$.
- Game $\mathbf{G}_1$: In this game, we simulate the ideal cipher oracles E and E^{-1}, by maintaining a list E-List, using a truly ideal cipher $\mathcal{E}$ and its inverse $\mathcal{E}^{-1}$, the signing oracle $\mathsf{Sign}_{\varphi^{-1}}(\cdot)$ with a list Σ-list, and the random oracles H-oracle and G-oracle, by maintaining the lists H-list and G-list, respectively.
  The simulation of the real attack game is described as follow:
  - $E_{(\cdot)}(\cdot)$-Oracle: A query $E_h(m_2 \parallel v)$ is answer by ω, where $\mathtt{Rule}\ \mathsf{EvalE}^{(1)}$: $\omega = \mathcal{E}_h(m_2 \parallel v)$, store the record $(h, (m_2 \parallel v), \omega)$ in E-list.

- $E^{-1}_{(\cdot)}(\cdot)$-Oracle: A query $E^{-1}_h(\omega)$ is answer by $m_2 \parallel v$, where
  $\texttt{Rule}\ \mathsf{InverE}^{(1)}$: $(m_2 \parallel v) = \mathcal{E}^{-1}_h(\omega)$, store the record $(h, (m_2 \parallel v), \omega)$ in E-list.
- G-Oracle: Answer to the query $G(\omega)$ is set as follow:
  $\texttt{Rule}\ \mathsf{G}^{(1)}$:
  1. if a record (ω, g) appears in G-List, the answer is g;
  2. otherwise the answer is randomly chosen from $\{0,1\}^{n-m}$ and the record (ω, g) is stored in G-list.
- H-Oracle: Answer to the query $H(m_1)$ is set as follow:
  $\texttt{Rule}\ \mathsf{H}^{(1)}$:
  1. if a record (m_1, h) appears in H-List, the answer is h;
  2. otherwise the answer is randomly chosen from $\{0,1\}^l$ and the record (m_1, h) is stored in H-list.
- $\mathsf{Sign}_{\varphi^{-1}}(\cdot)$-Oracle:
  $\texttt{Rule}\ \mathsf{S}^{(1)}$: For a signing query $\mathsf{Sign}_{\varphi^{-1}}(M)$, one first breaks up M as $(m_1, m_2) \in \{0,1\}^{n-m} \times \{0,1\}^{m-k}$ computes $b = \mathsf{PRF}_\theta(m_1, m_2)$, then asks for $h = H(m_1)$ to the H-oracle, then asks for $\omega = E_h(m_2 \parallel b^k)$ to $E_{(\cdot)}(\cdot)$-oracle, and then compute $s = G(\omega) \oplus m_1$. The signature σ is $\varphi^{-1}(s \parallel \omega)$. We store the $(m_1, m_2, b, h, \omega, g, \sigma)$ in Σ-list.
- $\mathsf{Ver}_\varphi(\cdot)$-Oracle:
  $\texttt{Rule}\ \mathsf{V}^{(1)}$: The game ends with the verification of the output σ from the adversary. One first computes $s \parallel \omega = \varphi(\sigma)$ where $s \in \{0,1\}^{n-m}$, $\omega \in \{0,1\}^m$ and then $m_1 = G(\omega) \oplus s$, then asks for $h = H(m_1)$ to the H-oracle, $(m_2 \parallel v) = E^{-1}_h(\omega) \in \{0,1\}^{m-k} \times \{0,1\}^k$ to the $E^{-1}_{(\cdot)}(\cdot)$-oracle, if $v = 0^k$ or $v = 1^k$, returns the message $M = (m_1, m_2)$; otherwise returns reject.

Now, we denote by Δ_j the statistical distance between the distribution of the adversary's view in the game G_j and in the game G_{j-1}. We see that the perfect simulation does not modify any probability in Game $\mathbf{G}_1$. Therefore, $\Delta_1 = 0$, and $\mathbf{Pr}[\mathsf{Forge}_0] = \mathbf{Pr}[\mathsf{Forge}_1]$.

– Game $\mathbf{G}_2$: In this game, we modify the simulations of the (random oracles) H-oracle and G-oracle in Game $\mathbf{G}_1$ as follows:
  - G-Oracle: Answer to the query $G(\omega)$ is set as follow:
    $\texttt{Rule}\ \mathsf{G}^{(2)}$:
    1. if a record (ω, g) appears in G-List, the answer is g;
    2. otherwise one randomly chooses g from $\{0,1\}^{n-m}$, if g has been set as $G(\omega')$ for some $\omega' \neq \omega$, aborts; else returns g as the answer and stores the record (ω, g) in G-list.
  - H-Oracle: Answer to the query $H(m_1)$ is set as follow:
    $\texttt{Rule}\ \mathsf{H}^{(2)}$:
    1. if a record (m_1, h) appears in H-List, the answer is h;
    2. otherwise one randomly chooses h from $\{0,1\}^l$, if h has been set as $H(m_1')$ for some $m_1' \neq m_1$, aborts; else returns g as the answer and stores the record (m_1, h) in H-list.

Let Δ_2 be the statistical distance between the distribution of the adversary's view in the game G_2 and in the game G_1. The simulations of the H-oracle and G-oracle in the game are indistinguishable from those in the previous game unless the simulations abort, which event is denoted Abort. The event Abort happens only when a collision on either G or H has occurred, which event is denoted Col-H-G. Hence,

$$\begin{aligned}\Delta_2 &= |\mathbf{Pr}[\mathsf{Forge}_1] - \mathbf{Pr}[\mathsf{Forge}_2]| \\ &= \mathbf{Pr}[\mathsf{Abort}] \\ &\leq \mathbf{Pr}[\mathsf{Col\text{-}H\text{-}G}] \\ &\leq \frac{(q_h + q_s + 1)^2}{2^{l+1}} + \frac{(q_g + q_s + 1)^2}{2^{n-m+1}}.\end{aligned} \tag{2}$$

– Game $\mathbf{G}_3$: In this game, we modify the simulation of the ideal cipher oracles E and E^{-1} in Game $\mathbf{G}_2$. The explicit rules of the simulations are set as follows:
  - $E_{(\cdot)}(\cdot)$-Oracle: When a query $E_h(m_2 \parallel v)$ [2] is submitted, we use the following rules:
    Rule $\mathsf{EvalE}^{(3)}$: Look for the record $(h, (m_2 \parallel v), \omega)$ in E-list:
    1. if the record is found, $E_h(m_2 \parallel v) = \omega$;
    2. otherwise randomly choose a $x \in \{0,1\}^n$, compute $(s \parallel \omega) = \varphi(x)$ (if it is the i-th query we may set $(s \parallel \omega) = y$), and then look for the record (m_1, h)[3] in H-list:
       (a) if the record (m_1, h) is found, compute $g = m_1 \oplus s$, if g has been set as $G(\omega')$ for some $\omega' \neq \omega$, stop (denoted by Stop1); otherwise return $E_h(m_2 \parallel v) = \omega$ and add (ω, g) to G-List.
       (b) otherwise when the record (m_1, h) isn't found, randomly choose $m_1 \in \{0,1\}^{n-m}$, if m_1 is asked to H before, stop (denoted by Stop2); else compute $g = m_1 \oplus s$, if g has been set as $G(\omega')$ for some $\omega' \neq \omega$, stop (denoted by Stop3); otherwise return $E_h(m_2 \parallel v) = \omega$, add (ω, g) to G-List and (m_1, h) to H-list.
       (c) store the record $(h, (m_2 \parallel v), \omega)$ in E-list and the record $(m_1, m_2, b, h, \omega, g, x)$ in Σ-list where $v = b^k$ (for the i-th query store the record $(m_1, m_2, b, h, \omega, g, \perp)$ in Σ-list).
  - $E^{-1}_{(\cdot)}(\cdot)$-Oracle: A query $E_h^{-1}(\omega)$ is answered according to the following rules:
    Rule $\mathsf{InverE}^{(3)}$: Look for the record $(h, (m_2 \parallel v), \omega)$ in E-list: if the record is found, return $m_2 \parallel v$; otherwise randomly choose $(m_2 \parallel \nu) \in \{0,1\}^{m-k} \times \{0,1\}^k$:
    1. if $\nu \in \{1^k, 0^k\}$, stop (denoted by Stop4);
    2. otherwise return $E_h^{-1}(\omega) = (m_2 \parallel \nu)$, store the record $(h, (m_2 \parallel \nu), \omega)$ in E-list.

[2] Here, we may assume $v = b^k$ in a query $E_h(m_2 \parallel v)$ where $b \in \{0,1\}$, otherwise the signature won't be valid.

[3] In Game $\mathbf{G}_2$, H has no collisions, then, we can find a unique m_1 as the preimage of a given h under H if h has been set.

There is no difference between GAME $\mathbf{G}_3$ and GAME $\mathbf{G}_2$ unless one of the events Stop1,Stop2, Stop3 and Stop4 happens. Therefore, the difference is the following probability

$$\mathbf{Pr}[\mathsf{Stop1} \vee \mathsf{Stop2} \vee \mathsf{Stop3} \vee \mathsf{Stop4}] \leq \mathbf{Pr}[\mathsf{Stop1}] + \mathbf{Pr}[\mathsf{Stop2}] + \mathbf{Pr}[\mathsf{Stop3}] + \mathbf{Pr}[\mathsf{Stop4}]. \quad (3)$$

Since there are only $q_g + q_s + 1$ queries to G, we can compute $\mathbf{Pr}[\mathsf{Stop1}] \leq \frac{q_E(q_g+q_s+1)}{2^{n-m}}$, where q_E is the number of the queries to $E_{(\cdot)}(\cdot)$-oracle. Similarly, we have $\mathbf{Pr}[\mathsf{Stop2}] \leq \frac{q_E(q_h+q_s+1)}{2^{n-m}}$ and $\mathbf{Pr}[\mathsf{Stop3}] \leq \frac{q_E(q_g+q_s+1)}{2^{n-m}}$. The probability that the event Stop4 happens is upper bounded by $\frac{2q_{E^{-1}}}{2^k}$ where $q_{E^{-1}}$ is the number of queries to $E^{-1}_{(\cdot)}(\cdot)$-oracle. Therefore,

$$\begin{aligned} \Delta_3 &= |\mathbf{Pr}[\mathsf{Forge}_2] - \mathbf{Pr}[\mathsf{Forge}_3]| \\ &\leq \frac{2q_E(q_g + q_s + 1) + q_E(q_h + q_s + 1)}{2^{n-m}} + \frac{2q_{E^{-1}}}{2^k} \\ &\leq \frac{3q_E(q_g + q_h + q_s + 1)}{2^{n-m}} + \frac{2q_{E^{-1}}}{2^k}. \end{aligned} \quad (4)$$

– GAME $\mathbf{G}_4$: In this game we shall use E-list, H-list, G-list and Σ-list to generate signatures instead of using the trapdoor φ^{-1} in the simulation of the signing oracle ($\mathsf{Sign}_{\varphi^{-1}}(\cdot)$-Oracle) of GAME $\mathbf{G}_3$. The rules of the simulation of $\mathsf{Sign}_{\varphi^{-1}}(\cdot)$-Oracle is described as follows:

- `Rule` $\mathsf{S}^{(1)}$: For a signing query $\mathsf{Sign}_{\varphi^{-1}}(M)$, one first breaks up M as $(m_1, m_2) \in \{0,1\}^{n-m} \times \{0,1\}^{m-k}$, randomly chooses one bit $b \in \{0,1\}$, then asks a query $h = H(m_1)$ to the H-oracle and a query $E_h(m_2 \parallel b^k)$ to $E_{(\cdot)}(\cdot)$-oracle, and then finds the record $(m_1, m_2, b, h, \omega, g, x)$ in Σ-list. Hence, the signature x satisfies $\varphi(x) = (s \parallel \omega)$ where $s = G(\omega) \oplus m_1$ according to the rules of $E_{(\cdot)}(\cdot)$-oracle in the previous game.

Except the i-th signature query, all the signature queries are answered by valid signatures. However, in this game one needs to submit at most q_s queries to G, H and $E_{(\cdot)}(\cdot)$ oracles respectively, which may lead to the abortion of the simulation. Exactly, whether such events will happen is the only difference between GAME $\mathbf{G}_4$ and GAME $\mathbf{G}_3$. If the events are denoted by AbortG, AbortH and AbortE respectively, Δ_4 is upper bounded by the probability of these events. Therefore :

$$\begin{aligned} \Delta_4 &= |\mathbf{Pr}[\mathsf{Forge}_3] - \mathbf{Pr}[\mathsf{Forge}_4]| \\ &\leq \mathbf{Pr}[\mathsf{AbortG}] + \mathbf{Pr}[\mathsf{AbortH}] + \mathbf{Pr}[\mathsf{AbortE}] \\ &\leq \frac{q_s(q_g + q_s + 1)}{2^{n-m}} + \frac{q_s \cdot (q_h + q_s + 1)}{2^l} + \frac{3q_s(q_h + q_g + q_s + 1)}{2^{n-m}} \\ &\leq \frac{q_s \cdot (q_h + q_s + 1)}{2^l} + \frac{4q_s(q_h + q_g + q_s + 1)}{2^{n-m}}. \end{aligned} \quad (5)$$

– GAME $\mathbf{G}_5$: This is the last game. Finally the forger will return a valid signature σ on message (m_1, m_2). We have $\varphi(\sigma) = s \parallel \omega$, $s \in \{0,1\}^{n-m}$,

$\omega \in \{0,1\}^m$ and $m_1 = G(\omega) \oplus s$. We ask for $h = H(m_1)$ to the H-oracle, $(m_2 \parallel v) = E_h^{-1}(\omega) \in \{0,1\}^{m-k} \times \{0,1\}^k$ to the $E_{(\cdot)}^{-1}(\cdot)$-oracle. Hence $v = b'^k$ where $b' \in \{0,1\}$. The event that $E_h(m_2 \parallel b'^k)$ is the i-th query to $E_{(\cdot)}(\cdot)$-oracle which is denoted by $\mathsf{GoodGuess}$, happens with probability $\frac{1}{q_E+q_s+1}$. In this case, we shall know that $\varphi(\sigma) = y$ and the reduction succeeds which is denoted by $\mathsf{Success}$. Otherwise we abort the game. Hence, we have

$$\varepsilon' = \mathbf{Pr}[\mathsf{Success}] = \mathbf{Pr}[\mathsf{Forge}_4 \wedge \mathsf{GoodGuess}] = \frac{1}{q_E + q_s + 1}\mathbf{Pr}[\mathsf{Forge}_4]. \quad (6)$$

For all, we have

$$\varepsilon - (q_E + q_s + 1) \cdot \varepsilon' \leq \frac{q_{E^{-1}}}{2^{k-1}} + \frac{3(q_h + q_s + 1)^2}{2^{l+1}} + \frac{9(q_g + q_h + q_s + 1) \cdot (q_E + q_g + q_s + 1)}{2^{n-m+1}} \quad (7)$$

The running time t of the reduction includes the running time of $\mathcal{F}$ and is otherwise dominated by the computation of φ denoted T_φ performed for each query to the signing oracle and the ideal cipher oracle $(E_{(\cdot)}(\cdot))$ at most. Therefore, we have $t' \leq t + (q_E + q_s + 1)T_\varphi$ □

If the underlying trapdoor permutation is induced by a claw free permutation, we can use the idea of the Katz-Wang construction [17] to achieve tight security in signature. More precisely, we only need to modify the simulation of the signing oracle and the ideal cipher oracle in the previous proof, then the following theorem holds.

Theorem 2. *Let $\mathcal{F}$ be adaptively chosen-message (to the signing oracle) adversaries, against the signature scheme. Let us assume that $\mathcal{F}$ can produce an existential forgery according to the definition 2, with success probability ε (within a time bound t, after $q_e = q_E + q_{E^{-1}}$, q_s, q_h and q_g queries to the keyed permutation oracles (which includes q_E forward queries and $q_{E^{-1}}$ inverse queries), the signing oracle, H-oracle and G-oracle, respectively). If the function φ is induced by a claw-free permutation, then we can inverse φ in time t', with probability ε' where $t' \leq t + (q_E + q_s + 1)T_\varphi$ and*

$$\varepsilon' \geq \frac{1}{2}(\varepsilon - \frac{q_{E^{-1}}}{2^{k-1}} - \frac{3(q_h + q_s + 1)^2}{2^{l+1}} - \frac{9(q_g + q_h + q_s + 1) \cdot (q_E + q_g + q_s + 1)}{2^{n-m+1}}). \quad (8)$$

3.3 How to Sign Long Messages

The padding in previous subsection for signature schemes only allows one to sign messages of length $n - k$. To sign a message m of arbitrary length greater than $n - k$, the message is split $m = (m_0, m_1, m_2) \in \mathfrak{M} = \{0,1\}^* \times \{0,1\}^{n-m} \times \{0,1\}^{m-k}$. m_1, m_2 will be recovered from the signed message. While m_0 will be transmitted in the clear. Details of the signature scheme are described as follows:

Key Generation Algorithm Gen: runs $\mathsf{Tp\text{-}Gen}(1^n)$ to obtain a pair of trapdoor permutation φ and φ^{-1}. Sets (φ, φ^{-1}) as the public key and secret key, respectively.

Signature Algorithm Sign: The space of the messages is $\mathfrak{M} = \{0,1\}^* \times \{0,1\}^{n-m} \times \{0,1\}^{m-k}$, the signature algorithm outputs a signature $(m_0 \parallel \sigma)$ into $\{0,1\}^* \times \{0,1\}^n$: on a message $M = (m_0, m_1, m_2) \in \mathfrak{M}$, one computes $b = \mathsf{PRF}_\theta(M)$, $h = H(m_0, m_1)$, $\omega = E_h(m_2 \parallel b^k)$, $s = G(\omega) \oplus m_1$ and then $\sigma = \varphi^{-1}(s \parallel \omega)$, returns $(m_0 \parallel \sigma)$.

Verification Algorithm Ver: On a signature $(m_0 \parallel \sigma)$, one first computes $s \parallel \omega = \varphi(\sigma)$ where $s \in \{0,1\}^{n-m}$, $\omega \in \{0,1\}^m$ and then $m_1 = G(\omega) \oplus s$, $h = H(m_0, m_1)$, $(m_2 \parallel v) = E_h^{-1}(\omega) \in \{0,1\}^{m-k} \times \{0,1\}^k$. Finally, if $v = 0^k$ or $v = 1^k$, the verifier returns the message $M = (m_0, m_1, m_2)$; otherwise returns reject.

Therefore, the padding for signature schemes can be used to sign on long messages. More precisely, part of the message can be recovered from the signed message. The bandwidth in communication can be optimal if we use this kind of padding for signature scheme to sign long messages.

Meanwhile, the security proof still holds if all answers to oracle queries are independent for different values of (m_0, m_1) and we can obtain the same result within a similar theorem.

4 Parameters and Discussions

For practical purpose, n and m is much greater than the bit-size of the redundancy k (where $m \leq 256$ since the existing largest block cipher can't be directly used to encrypt a plaintext with block size greater than 256 bits [16]). Since all the queries to the oracles is upper bounded, we may assume that $1 + q_s + q_h + q_g + q_E + q_{E^{-1}} \leq Q$. Then, without loss of generosity, assuming that $Q \leq t' \approx t \leq t + Q \leq 2^k$, the quantity $Q \cdot 2^{-l}$, $Q \cdot 2^{m-n}$, or even $Q^2 \cdot 2^{-l}$, $Q^2 \cdot 2^{m-n}$ can be ignored in front of $Q \cdot 2^{-k}$ if we assume $l > 3k$ and $n - m > 3k$. Therefore, the above reduction cost provides that

$$\begin{aligned} \frac{\varepsilon}{t} &\leq \frac{Q\varepsilon'}{t} + \frac{2}{2^k}, \quad \text{in the general case;} \\ \frac{\varepsilon}{t} &\leq \frac{2\varepsilon'}{t} + \frac{2}{2^k}, \quad \text{if } \varphi \text{ is induced by a claw-free permutation.} \end{aligned} \tag{9}$$

In the general case, if the underlying trapdoor permutation is $2k$-bit secure, we shall get a signature scheme which is nearly k-bit secure with only k bits of redundancy (or message expansion). According to the first theorem in [13], our padding leads to an optimal signature. Due to the fact that $\frac{\varepsilon}{t} \leq \frac{Q\varepsilon'}{t} + \frac{2}{2^k}$, we only require that $\frac{\varepsilon'}{t} \leq 2^{-2k}$, then we shall roughly get $\frac{\varepsilon}{t} \leq c \cdot 2^{-k}$ where c is a small constant.

If φ is induced by a claw-free permutation, the padding not only achieves the lower bounds of message expansion for signature schemes (optimal for the length of message recovery), but also leads to a tight security reduction. Namely, we needn't require the underlying permutation to be $2k$-bit secure. For a tight security reduction implies that the signature scheme and underlying trapdoor

permutation have the same security level, the padding for signature schemes is also optimal for the security of theoretical design.

With a usual setting of the security bound, in the case that φ is induced by a claw-free permutation, $k = 80$ is secure enough in the real world.

4.1 How to Implement in the Real World

As we have mentioned previously, the padding for signature with message recovery is also practical and efficient in the real world. Here, we explain the details on how to implement such a padding for signature schemes using the existing cryptographical primitives. Since the most popular trapdoor (one-way) permutations are the RSA permutations we can use the RSA permutations as the underlying permutations. Luckily, the RSA permutations are claw-free as well. Therefore, the RSA permutations are widely used in all kinds of cryptographical standards, such as the standards: RSA-PSSR, RSA-OAEP, etc.

Our theorem shows that the padding for signature schemes leads to a tight security. Hence, for the particular case of RSA, we can use a 1024-bit modulus. Due to the Lenstra-Verheul's estimation [15], if RSA is (t', ε')-secure, then $t'/\varepsilon' \geq 2^{80}$. Therefore, the parameters are suggest to be $n = 1024$, $k = 81$ and $m = l = 256 > 3k = 243$, then we shall get a signature with 80-bit security level.

We shall use two hash functions $H : \{0,1\}^* \to \{0,1\}^{256}$ and $G : \{0,1\}^{256} \to \{0,1\}^{768}$ which can be viewed as random oracles. For the ideal cipher, we may choose the block cipher, NUSH, or a variant of the NIST AES standard, 256-bit plaintext block Rijndael, with a 256-bit key over 14 rounds[16], to instantiate the keyed permutation $E(\cdot) : \{0,1\}^l \times \{0,1\}^m \to \{0,1\}^m$. Both the key length and the block size can be 256 bits.

4.2 Comparison with Some Other Schemes

Various practical signature schemes (with message recovery) were proposed through the last few years. However, most of them are proven secure in the random oracle model. None of the schemes proven secure in the random oracle model can achieve optimality in bandwidth including the RSA-PSSR, Katz-Wang's improved PSSR.

Indeed, the Katz-Wang's paper [17] that proves that PSS with a one-bit salt is sufficient when the underlying permutation is induced by a claw-free permutation. However, it is not optimal. From theorem 3 in [17] and the suggestion in [3], we get that the expansion (length of redundancy and randomness), in other word, the difference between the length of the signed message and the original message should be nearly $k_1 + 1 = 180$ bits if taking $Q = 2^{60}$, $\varepsilon' = 2^{-60}$. With the same assumption that $Q = 2^{60}$, $\varepsilon' = 2^{-60}$, we may see that our padding for signature schemes only needs nearly $k = 120$ bits of expansion. More exactly, Katz-Wang's construction reduces the randomness to one bit, but keeps the original redundancy of PSS. While, our padding for signature schemes bounds the length of redundancy and randomness as small as possible. Therefore, Katz-Wang's improved PSS is not optimal in bandwidth, but ours is optimal. To our

knowledge, the problem on how to construct an optimal signature scheme in the random oracle model still remains open now.

On the other hand, all the optimal signature schemes (with message recovery) depend on either the random permutations or the ideal ciphers. However, these permutations (including the ideal ciphers) all require large block sizes (i.e., block sizes larger than the block size of a cipher such as AES, etc)[13,17,4]. Therefore, these proposals are less efficient in the real world due to the cause that larger domain cipher calls the underlying block cipher many times[14] when it enciphers the message (or deciphers the ciphertext).

We compare our scheme (denoted by Σ_{POPSS}) with practical signature schemes: Katz-Wang's improved PSSR[17], the signature scheme from OAEP3r [4](denoted by Σ_{OAEP3r}), and the short signature schemes in [5,6]. Meanwhile, we compare our scheme with all the previous proposed optimal signature schemes: Granboulan's proposal [13](denoted by Σ_{OPSSR}), the signature scheme from the universal padding OPbP in [4] (denoted by Σ_{OPbP}) and Katz-Wang's scheme in [17]. Let TP and CFP denotes trapdoor permutations and claw-free permutations, respectively. CDHP denotes the computational Diffie-Hellman problem in Gap of Diffie-Hellman Problems (GDH) groups. Details are showed in table 1.

Table 1. Comparison with Some Other Schemes

Signature Schemes:	Message Recovery	Tight Security	Assumption	Optimal	Efficient
Improved RSA-PSSR in[17]	yes	yes	RSA	no	yes
Σ_{OAEP3r}	yes	no/yes	TP/CFP	no	yes
Short signatures in [5,6]	no	no	CDHP	no	yes
Optimal Signature Schemes:					
Σ_{OPSSR}	yes	no	TP	yes	no
Katz and Wang's Proposal in[17]	yes	yes	CFP	yes	no
Σ_{OPbP}	yes	yes	TP/CFP	yes	no
Σ_{POPSS}	yes	yes	TP/CFP	yes	yes

5 Conclusions

This paper proposes a practical, efficient optimal padding for signature schemes with message recovery. The padding is simple and efficient only employing two random oracles and an ideal cipher with a practical small block size. We provide a perfect solution to the problem on whether there exists a practical optimal padding for signature schemes with message recovery that can be efficiently implemented in the real world. Exactly, we've designed practical, efficient optimal signature schemes by replacing the idealized oracles with a true block cipher and two hash functions. We show that the padding is provable secure if the underlying trapdoor permutation is secure. The padding for signature schemes leads the scheme to achieve a tight security reduction when the underlying trapdoor permutation is induced by a claw free permutation.

Particularly, in the case of RSA, the padding makes the signature scheme achieve not only optimality in bandwidth but also a tight security.

Acknowledgments

We would like to thank Zhenfu Cao, Xuejia Lai and the three reviewers of CT-RSA 2007 for their careful reading and valuable comments on earlier drafts of the paper. The third author gratefully acknowledges the support by the NSFC grant No. 44010860, and the Doctoral Program Foundation of EMC No. 20060081.

References

1. J. H. An, Y. Dodis, and T. Rabin.: On the security of joint signature and encryption. In: Advances in Cryptology-EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pages 83–107. Springer-Verlag, Berlin Heidelberg, New York, 2002.
2. M. Bellare, P. Rogaway.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on Computer and communications Security, pages 62–73. ACM Press, New York, 1993.
3. M. Bellare, P. Rogaway.: The exact security of digital signatures –how to sign with RSA and Rabin. In: Advances in Cryptology-EUROCRYPT'96. Lecture Notes in Computer Science, Vol.1070, pages 399–416. Springer-Verlag, Berlin Heidelberg, New York, 1996.
4. B. Chevallier-Mames, D. H. Phan, and D. Pointcheval.: Optimal Asymmetric Encryption and Signature Paddings. In: Proceedings of the Conference on Applied Cryptography and Network Security (ACNS'05), LNCS 3531, Pages 254–268. Springer-Verlag, Berlin, 2005.
5. D. Boneh, B. Lynn, and H. Shacham.: Short signatures from the Weil pairing. In: Advances in Cryptology-Asiacrypt'2001. Lecture Notes in Computer Science, Vol. 2248, pages 514–532. Springer-Verlag, Berlin Heidelberg, New York, 2001.
6. D. Boneh and X. Boyen.: Short signatures without random oracles. In: Proceedings of EUROCRYPT 2004, LNCS 3027, pages. 56–73. Springer-Verlag, Berlin Heidelberg, 2004.
7. R. Canetti, O. Goldreich, and S. Halevi.: The random oracle methodology, revisited. In: Proceedings of 30th Annual ACM Symposium on Theory of Computing (STOC), pages 209–218. ACM press, New York, 1998.
8. J. S. Coron.: On the exact security of full domain hash. In: Prooceedings of CRYPTO'2000. Lecture Notes in Computer Science, Vol. 1880, pages 229–235. Springer-Verlag, Berlin Heidelberg, New York, 2000.
9. J. S. Coron.: Optimal Security Proofs for PSS and Other Signature Schemes. In: Advances in Cryptology-EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pages 272–287. Springer-Verlag, Berlin Heidelberg, New York, 2002.
10. Y. Dodis, L. Reyzin.: On the Power of Claw-Free Permutations. In: Third International Conference, SCN 2002. (Amalfi, Italy, September 11-13, 2002), Lecture Notes in Computer Science, Vol. 2576, pages 55–73. Springer-Verlag, Berlin Heidelberg, New York, 2003.
11. E. J. Goh, S. Jarecki.: A signature scheme as secure as the Diffie-Hellman problem. In: Advances in Cryptology-EUROCRYPT 2003. Lecture Notes in Computer Science, vol. 2656, pages 401–415. Springer-Verlag, Berlin Heidelberg, New York, 2003.
12. S. Goldwasser, S. Micali, and R. Rivest.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal of Computing, 17(2):281–308, April 1988.

13. L. Granboulan.: Short signatures in the random oracle model. In: Proceedings of Asiacrypt'02, volume 2501 of LNCS, pages 364–378, Springer-Verlag, Berlin, 2002.
14. S. Halevi, P. Rogaway.: A Tweakable Enciphering Mode. Advances in Cryptology - CRYPTO '03, Lecture Notes in Computer Science, vol. 2729, pp. 482-499, Springer-Verlag, 2003.
15. A. Lenstra and E. Verheul.: Selecting Cryptographic Key Sizes. In PKC'00, LNCS 1751, pages 446–465. Springer-Verlag, Berlin, 2000.
16. NESSIE consortium, "NESSIE Security report". Deliverable Type Report D21, NESSIE, 2002. Available from `http://www.cosic.esat.kuleuven.ac.be/nessie/index.html`.
17. J. Katz, N. Wang.: Efficiency improvements for signature schemes with tight security reductions. In: Proceedings of the 10th ACM conference on Computer and communication security, ACM Press, pages 155–164. New York, USA, 2003.
18. N. Koblitz, A. Menezes.: Another look at "provable security". Cryptology ePrint Archive, Report 2004/152 (2004). Available from `http://www.cacr.math.uwaterloo.ca/~ajmeneze/publications/provable.pdf`
19. B. Libert, J. J. Quisquater.: The Exact Security of an Identity Based Signature and its Applications. Cryptology ePrint Archive, Report 2004/102. Available from `http://eprint.iacr.org/`.
20. D. Pointcheval, J. Stern.: Security proofs for signature schemes. In: Advances in Cryptology-EUROCRYPT'96. Lecture Notes in Computer Science, vol. 1070, pages 387–398. Springer-Verlag, Berlin Heidelberg, New York, 1996.
21. L. Pintsov and S. Vanstone.: Postal revenue collection in the digital age. In: Proceedings of Financial Cryptography 2000, volume 1962 of LNCS, pages 105–20. Springer-Verlag, Berlin, 2000.
22. R. L. Rivest, A. Shamir, and L. M. Adleman.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2): 120–126, 1978.

Directed Transitive Signature Scheme

Xun Yi

School of Computer Science and Mathematics
Victoria University, PO Box 14428, Melbourne City MC
Victoria 8001, Australia

Abstract. In 2002, Micali and Rivest raised an open problem as to whether directed transitive signatures exist or not. In 2003, Hohenberger formalized the necessary mathematical criteria for generic directed transitive signature scheme, showing that the edge signatures in such a scheme form a special (and powerful) mathematical group, called Abelian trapdoor group with infeasible inversion, which is not known to exist. In this paper, we consider a directed graph whose transitive reduction is a directed tree, on which we propose a natural RSA-based directed transitive signature scheme $\mathcal{RSADTS}$. In this particular case, we have answered the open problem raised by Micali and Rivest. We have proved that $\mathcal{RSADTS}$, associated to a standard digital signature scheme, is transitively unforgeable under adaptive chosen-message attack if the RSA inversion problem over a cyclic group is hard and the standard digital signature is secure. Furthermore, $\mathcal{RSADTS}$ has even better performance than $\mathcal{RSATS}$-1 in certain circumstance.

Keywords: Directed transitive signature, transitive closure and reduction, RSA inversion problem over a cyclic group.

1 Introduction

The concept of transitive signature was envisioned by Micali and Rivest [1] in 2002. Transitive signature aims to authenticate the transitive closure of a dynamically growing graph $G = (V, E)$, denoted as $\tilde{G} = (V, \tilde{E})$, in which there exists an edge (i, j) in $\tilde{E}$ if there exists a path from nodes i to j, where $i, j \in V$. The original idea of transitive signature is that the signer, having secret key tsk and public key tpk, is able to sign any node and any edge of G such that given signatures on nodes i, j, k in V, and signatures on edges (i, j), (j, k) in $\tilde{E}$, anyone in possession of tpk can compose a signature on the edge (i, k) in $\tilde{E}$. However, without tsk, it is hard to create a valid signature of an edge or a node outside $\tilde{G}$.

As suggested by Micali and Rivest [1], transitive signature for an undirected graph can be used to authenticate administrative domains, where nodes stand for machines and an undirected edge (i, j) means that i and j are in the same domain, while transitive signature for a directed graph can be used to authenticate a military chain-of-command, where nodes stand for personnel and a directed edge (i, j) from nodes i to j means that i commands (or controls) j.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 129–144, 2007.

Two transitive signature schemes, $\mathcal{DLTS}$ and $\mathcal{RSATS}$-1, were firstly proposed by Micali and Rivest [1]. Shortly afterwards, Bellare and Neven [2][3] proposed a series of transitive schemes, such as $\mathcal{FactTS}$-1, $\mathcal{DLTS}$-1$\mathcal{M}$, $\mathcal{GapTS}$-1, $\mathcal{RSATS}$-2, $\mathcal{FactTS}$-2, and $\mathcal{GapTS}$-2. $\mathcal{DLTS}$ is similar to Okamoto's ID scheme using two generators [4] while $\mathcal{DLTS}$-1$\mathcal{M}$ is similar to Schnorr's ID scheme using one generator [5]. Recently, Shahandashti et al. [6] proposed a short transitive signature scheme based on bilinear maps that is the same as $\mathcal{GapTS}$-1 scheme.

In [1], Micali and Rivest proved that $\mathcal{DLTS}$ is transitively unforgeable under adaptive chosen-message attack assuming that the discrete logarithm problem is hard in an underlying prime-order group and an underlying standard signature scheme is secure. They pointed out that even though the natural RSA based transitive signature scheme $\mathcal{RSATS}$-1 can be proved to be transitively unforgeable under nonadaptive chosen-message attack, there is no known proof of transitively unforgeable under adaptive chosen-message attack.

In [2][3], Bellare and Neven proved $\mathcal{RSATS}$-1 to be secure (transitively unforgeable under adaptive chosen-message attack), under the assumption that the one-more RSA inversion problem is hard and the underlying standard digital signature $\mathcal{SDS}$ scheme is secure [7] (unforgeable under adaptive chosen-message attack). One-more RSA inversion problem was introduced by Bellare et al. [8] in order to prove the security of Chaum's blind signature scheme [9]. It was also used in [10] to prove security of Guillou-Quisquater (GQ) identification scheme [11] against the impersonation attack.

Bellare and Neven also proved that (1) $\mathcal{FactTS}$-1 is secure if the factoring problem is hard and the underlying $\mathcal{SDS}$ is secure; (2) $\mathcal{DLTS}$-1$\mathcal{M}$ is secure if one-more discrete logarithm problem [10] is hard and the underlying $\mathcal{SDS}$ is secure; and (3) $\mathcal{GapTS}$-1 is secure if one-more gap Diffie-Hellman problem [12][13] is hard and the underlying $\mathcal{SDS}$ is secure.

$\mathcal{DLTS}$, $\mathcal{RSATS}$-1, $\mathcal{FactTS}$-1, $\mathcal{DLTS}$-1$\mathcal{M}$, and $\mathcal{GapTS}$-1 [2][3] follow the node certificate paradigm, in which: (1) The signer associates to each node i in the current graph a node certificate consisting of a public label $L(i)$ and a signature on the concatenation of i and $L(i)$ under the standard signature scheme, and creates the signature of an edge including the certificates of its endpoints plus an edge label δ; (2) Verification of an edge signature involves relating the edge label to the public labels of its endpoints as provided in the node certificates and verifying the standard signatures in the node certificates; (3) Composition involves algebraic manipulation of edge labels.

$\mathcal{RSATS}$-2, $\mathcal{FactTS}$-2, and $\mathcal{GapTS}$-2 [2][3] eliminate node certificates by specifying the public label of a node i as the output of a hash function applied to i. No explicit certification is attached to this value. In [2][3], the edge label is shown to provide an "implicit authentication" of the associated node label and $\mathcal{RSATS}$-2, $\mathcal{FactTS}$-2, and $\mathcal{GapTS}$-2 are proved to be transitively unforgeable under adaptive chosen-message attack, in a model where the hash function is a random oracle [14]. Therefore, the standard signature scheme and all associated costs are removed.

The above transitive signature schemes are designed for undirected graphs, in which (i, j) and (j, i) stand for the same edge and therefore they have the same edge signature. A transitive signature scheme for undirected graphs cannot be used to authenticate directed graphs, in which (i, j) and (j, i) stand for distinct edges and thereby they have distinct edge signatures.

In 2002, Micali and Rivest raised an open problem [1]: "The problem of finding a directed transitive signature scheme remains a very interesting open problem. We have not been able to make much progress on this problem." In general, a directed transitive signature DTS scheme allows the signer to sign a subset of edges on a directed graph in such a way that anyone can compose the signatures on edges (i, j) and (j, k) to obtain the signature on (i, k).

In 2003, Hohenberger [15] formalized the necessary mathematical criteria for generic DTS scheme when the signatures can be composed in any order, showing that the edge signatures in such a scheme form a special (and powerful) mathematical group, called Abelian trapdoor group with infeasible inversion (ATGII), which is not known to exist. Such a group would only be possible when the order of the group remains secret [16]. Furthermore, a DTS scheme is more complex - in a black box sense - than standard signature, public key encryption and oblivious transfer, and a pseudo-free ATGII is sufficient for a secure DTS construction.

Kuwakado and Tanaka [17] constructed a transitive signature scheme for directed trees in 2003. However, Yi et al. [18] has shown that it is insecure against a forgery attack, in which directed edge signatures can be forged by composing the existing directed edge signatures provided by the signer.

Hohenberger's criteria for a generic DTS scheme is applicable for general directed graphs. In a special case where the transitive reduction of a directed graph is a directed tree, can we find a directed transitive signature scheme on it?

In this paper, we construct a natural RSA based directed transitive signature $\mathcal{RSADTS}$ scheme for a directed graph whose transitive reduction is a directed tree. Our basic idea is that a node i is mapped to an element $L(i)$ in a cyclic subgroup of Z_n^* for an RSA modulus n, and a directed edge (i, j) is mapped to odd prime δ_{ij}, such that $L(i)^{\delta_{ij}} = L(j)(mod\ n)$.

The main contributions of this paper include: (1) By $\mathcal{RSADTS}$ scheme, we have answered the open problem raised by Micali and Rivest as to whether a directed transitive signature scheme exists or not in the case where the transitive reduction of a directed graph is a directed tree; (2) We slightly modify the definitions of a UTS scheme, its correctness and security given by Bellare and Neven [2][3] to fit into a $\mathcal{RSADTS}$ scheme; (3) We formally define the RSA inversion problem over a cyclic group; (4) We prove $\mathcal{RSADTS}$ to be transitively unforgeable under adaptive chosen-message attack if the RSA inversion problem over a cyclic group is hard and the associated standard signature scheme is unforgeable under adaptive chosen-message attack; (5) We find that $\mathcal{RSADTS}$ has better performance than $\mathcal{RSATS}$-1 in certain circumstance.

The rest of this paper is organized as follows: Section 2 introduces notations and definitions; Section 3 presents $\mathcal{RSADTS}$ scheme; Section 4 gives the security

proof of $\mathcal{RSADTS}$ scheme; Section 5 discusses the performance of $\mathcal{RSADTS}$ scheme; Conclusions are drawn in the last section.

2 Notations and Definitions

Notations: The notation $x \xleftarrow{R} S$ denotes that x is randomly selected from the set S. Let $\mathcal{N} = \{1, 2, \cdots, n, \cdots\}$ and $\mathcal{P}$ stand for the set of all odd primes, $\emptyset$ represent the empty set, $\|$ the concatenation operator on strings, $|S|$ the order of a set S, and $<\mathcal{G}>$ a cyclic subgroup of Z_n^* generated by an integer $\mathcal{G}$, where n is a product of two safe primes $p = 2p' + 1$ and $q = 2q' + 1$, such that p' and q' are also primes. If A is a possible randomized algorithm, then the notation $x \leftarrow \mathsf{A}(a_1, a_2, \cdots, a_n)$ denotes that x is assigned the outcome of the experiment of running A on inputs $a_1, a_2, \cdots, a_n$.

Graph: In this paper, we consider a directed graph $G = (V, E)$, whose transitive reduction is a directed tree, and work on its transitive closure. The transitive closure, denoted as $\tilde{G} = (\tilde{V}, \tilde{E})$, is defined to have $\tilde{V} = V$ and to have an edge (i, j) in $\tilde{E}$ if and only if there is a path from nodes i to j in G. The transitive reduction, denoted as $G^* = (V^*, E^*)$, is defined to be the minimum graph with the same transitive closure as G. It is obvious that $V^* = V$. In a directed graph, each directed edge is associated with an ordered pair of nodes (i, j), where i is the initial node and j the terminal node, and thus (i, j) and (j, i) stands for distinct directed edges. A directed tree is a directed graph which is a tree if the directions on the edges are ignored. A tree has some properties as follows:

- If it has $|V|$ nodes, then it has exactly $|V| - 1$ edges.
- There is exactly one path between every pair of nodes.
- If any two of nodes which are not adjacent are joined directly by an edge, then the resulting graph possesses exactly one cycle.

Directed Transitive Signature (DTS*) Scheme:* A directed transitive signature scheme $\mathcal{DTS} = (\mathsf{TKG}, \mathsf{TSign}, \mathsf{TVf}, \mathsf{Comp})$ is defined by four polynomial-time algorithms as follows:

- The randomized key generation algorithm TKG takes 1^k as input, where $k \in \mathcal{N}$ is the security parameter, and returns a pair (tpk, tsk), where tpk is the public key while tsk the matching secret key.
- The signature algorithm TSign, which could be stateful or randomized (or both), takes inputs the secret key tsk and a directed edge (i, j), where $i, j \in \mathcal{N}$, and returns an original signature σ_{ij} of (i, j) relative to tsk. If stateful, it maintains state which it updates upon each invocation.
- The deterministic verification algorithm TVf, given tpk, a directed edge (i, j), and a candidate signature σ_{ij}, returns either 1 or 0. If the output is 1, σ_{ij} is said to be a valid signature of (i, j) relative to tpk.
- The deterministic composition algorithm Comp takes tpk, two directed edges (i, j) and (j, k), and two signatures σ_{ij} and σ_{jk} as inputs, and returns either a composed signature σ_{ik} of edge (i, k) or $\perp$ to indicate failure.

In practice, it is desirable to allow users to name nodes with whatever identifiers they choose, but these names can always be encoded as integers [2]. We assume that the nodes of the graph are positive integers.

Correctness of DTS *Scheme:* Naturally, it is required that if σ_{ij} is an original signature of directed edge (i, j) relative to tsk then it is a valid signature of (i, j) relative to tpk. A transitive signature scheme allows to compose a signature σ_{ik} with two signatures σ_{ij} and σ_{ik}. Therefore, a signature is legitimate if it is either obtained by the signer, or obtained by applying the composition algorithm to legitimate signatures.

The formal definition of correctness takes into account the statefulness and associates to any algorithm A (deterministic, halting, but not computationally limited) and security parameter $k \in \mathcal{N}$ the experiment shown in Fig. 1, which provides A with oracles $\mathsf{TSign}(tsk, \cdot, \cdot)$ and $\mathsf{Comp}(tpk, \cdot, \cdot, \cdot, \cdot, \cdot)$, where tpk, tsk have been produced by running TKG on input 1^k. In this experiment, the TSign oracle maintains state and update this state each time it is invoked.

$(tpk, tsk) \leftarrow \mathsf{TKG}(1^k)$
$S \leftarrow \emptyset, Legit \leftarrow true, NotOK \leftarrow false$
Run A with its oracles until it halts, replying to its oracle queries as follows:
If A makes TSign query on (i, j) then
  If $[(i = j) \vee (\{i, j\} \in V)]$ then $Legit \leftarrow false$
  Else
    Let σ_{ij} be the output of the TSign oracle
    $S \leftarrow S \cup \{(i, j, \sigma_{ij})\}$
    If $\mathsf{TVf}(tpk, i, j, \sigma_{ij}) = 0$ then $NotOK \leftarrow true$
If A makes Comp query on $(i, j, k), \sigma_{ij}, \sigma_{jk}$ then
  If $[(i, j, k$ are not all distinct$) \vee ((i, j, \sigma_{ij}) \notin S) \vee ((j, k, \sigma_{jk}) \notin S)]$
  Then $Legit \leftarrow false$
  Else
    Let σ_{ik} be the output of the Comp oracle
    $S \leftarrow S \cup \{(i, k, \sigma_{ik})\}$
    If $(\mathsf{TVf}(tpk, i, k, \sigma_{ik}) = 0)$ then $NotOK \leftarrow true$
When A halts, it outputs $(Legit \wedge NotOK)$

Fig. 1. An experiment to define the correctness of a directed transitive signature scheme $\mathcal{DTS} = (\mathsf{TKG}, \mathsf{TSign}, \mathsf{TVf}, \mathsf{Comp})$

Definition 2.1. A directed transitive signature DTS scheme is said to be correct if for every algorithm A and every $k \in \mathcal{N}$, the output of the experiment of Fig. 1 is true with probability zero.

As A queries, the experiment computes a Boolean $Legit$ which is set to $false$ if A makes an "illegitimate" query, and a Boolean $NotOK$ which is set to $true$ if an invalid signature is returned by TSign or Comp oracles on a "legitimate" query. To win, A must stay legitimate (meaning $Legit = true$), but violate correctness (meaning $NotOK = true$). The experiment returns $true$ if and only if A wins. The definition needs that this happens with probability zero.

Different from the definition of correctness given in [2], we do not require the real and composed signatures to be the same or statistically indistinguishable. In fact, only one signature exists for each edge of the transitive closure, which is either produced by TSign or composed by Comp. The signer never produce a signature which can be composed by existing signatures.

Security of DTS *Scheme:* A forgery is a valid directed transitive signature on an edge not in the transitive closure. We associate $\mathcal{DTS} = (\mathsf{TKG}, \mathsf{TSign}, \mathsf{TVf}, \mathsf{Comp})$ to any algorithm F (called $\mathsf{dtu-cma}$ adversary) and security parameter $k \in \mathcal{N}$ the experiment $\mathbf{Exp}^{\mathsf{dtu-cma}}_{\mathcal{DTS},\mathsf{F}}(k)$ of Fig. 2, which provides F with input tpk and an oracle $\mathsf{TSign}(tsk, \cdot, \cdot)$. The oracle is assumed to maintain states.

$(tpk, tsk) \leftarrow \mathsf{TKG}(1^k)$
$S = \{(i, j, \sigma_{ij})\} \xleftarrow{R} \mathsf{TSign}(tsk, \cdot, \cdot)$
$(i', j', \sigma'_{i'j'}) \xleftarrow{R} \mathsf{F}(tpk, S)$
Let $E = \{(i, j) | \exists (i, j, \sigma_{ij}) \in S\}$, $V = \{i | (\exists (i, j) \in E) \vee (\exists (j, i) \in E)\}$
Let $G = (V, E)$, $\tilde{G} = (V, \tilde{E})$, $\tilde{S} = \{(i, j, \sigma_{ij}) | ((i, j) \in \tilde{E}) \wedge (\mathsf{TVf}(i, j, \sigma_{ij}) = 1)\}$
If $(i', j', \sigma'_{i'j'}) \in \tilde{S} \vee \mathsf{TVf}(i', j', \sigma'_{i'j'}) = 0$ then return 0
Else return 1

Fig. 2. An experiment to define the security of a directed transitive signature scheme $\mathcal{DTS} = (\mathsf{TKG}, \mathsf{TSign}, \mathsf{TVf}, \mathsf{Comp})$

The experiment $\mathbf{Exp}^{\mathsf{dtu-cma}}_{\mathcal{DTS},\mathsf{F}}(k)$ returns 1 if and only if F succeeds in producing at least one forgery. The advantage of F in its forgery attack on $\mathcal{DTS}$ is defined as

$$\mathbf{Adv}^{\mathsf{dtu-cma}}_{\mathcal{DTS},\mathsf{F}}(k) = Pr[\mathbf{Exp}^{\mathsf{dtu-cma}}_{\mathcal{DTS},\mathsf{F}}(k) = 1] \tag{1}$$

for $k \in \mathcal{N}$, where the probability is taken over all the random choices made in the experiment.

Definition 2.2. A directed transitive signature scheme $\mathcal{DTS} = (\mathsf{TKG}, \mathsf{TSign}, \mathsf{TVf}, \mathsf{Comp})$ is said to be transitively unforgeable under adaptive chosen-message attack if the function $\mathbf{Adv}^{\mathsf{dtu-cma}}_{\mathcal{DTS},\mathsf{F}}(k)$ is negligible for any adversary F whose running time is polynomial in the security parameter k.

Standard Digital Signature (SDS*) Scheme:* Our construction will use an underlying standard digital signature scheme $\mathcal{SDS} = (\mathsf{SKG}, \mathsf{SSign}, \mathsf{SVf})$, described as

usual via its polynomial time key generation (SKG), signing (SSign), and verification (SVf) algorithms. Based on the security definition of unforgeability under chosen-messages attack [7], a forger is given adaptive oracle access to the signing algorithm, meaning the forger can choose the next query based on the oracle's answer to the previous one, and its advantage $\mathbf{Adv}^{\mathsf{uf-cma}}_{\mathcal{SDS},\mathsf{B}}(k)$ in breaking $\mathcal{SDS}$ is defined as the probability that it outputs a valid signature for a message that was not one of its previous oracle queries. The scheme $\mathcal{SDS}$ is said to be unforgeable under adaptive chosen-message attack if $\mathbf{Adv}^{\mathsf{uf-cma}}_{\mathcal{SDS},\mathsf{B}}(k)$ is negligible for every polynomial-time forger.

3 Directed Transitive Signature Scheme

In this section, we propose a natural RSA based directed transitive signature scheme $\mathcal{RSADTS}$ for a directed graph $G = (V, E)$ whose transitive reduction $G^* = (V, E^*)$ is a directed tree.

Associated to a RSA-based cyclic group generator $\mathsf{K}_{\mathsf{rsacg}}$ and any standard signature scheme $\mathcal{SDS} = (\mathsf{SKG}, \mathsf{SSign}, \mathsf{SVf})$, a directed transitive signature scheme $\mathcal{RSADTS} = (\mathsf{TKG}, \mathsf{TSign}, \mathsf{TVf}, \mathsf{Comp})$ is defined as follows.

1. $\mathsf{TKG}(1^k)$ runs as follows:
   (1.1) Run $\mathsf{SKG}(1^k)$ to generate a key pair (spk, ssk).
   (1.2) Run $\mathsf{K}_{\mathsf{rsacg}}(1^k)$ to produce a triple $(<\mathcal{G}>, n, \varphi(n))$, where $n = pq$, p, q are two safe primes, $2^{k-1} < n < 2^k$, $\varphi(n) = (p-1)(q-1)$, and $<\mathcal{G}>$ is a cyclic subgroup of Z_n^* generated by an integer $\mathcal{G}$ such that $\mathcal{G}^2 \neq 1 (mod\ n)$.
   (1.3) Output $tpk = (<\mathcal{G}>, n, spk)$ as the public key and $tsk = (\varphi(n), ssk)$ as the secret key.
2. The signing algorithm TSign maintains state (V, Δ, L, Σ) where $V \subseteq \mathcal{N}$ is the set of all queried nodes, the function L: $V \rightarrow <\mathcal{G}>$ assigns to each node $i \in V$ a *public label* $L(i)$, while the function Δ: $V \rightarrow \mathcal{P}$ assigns to each edge $(i, j) \in E^*$ a *public label* δ_{ij}, and the function Σ: $V \rightarrow \{0,1\}^*$ assigns to each node i a standard digital signature $\Sigma(i)$ on $i \| L(i)$ under ssk with SSign. The node certificate of node i is $C_i = (i, L(i), \Sigma(i))$.

   Choosing a node r as a reference node, when invoked on inputs $tsk = (\varphi(n), ssk)$ and an edge $(i, j) \in E^*$, meaning when asked to produce a signature on the edge $(i, j) \in E^*$, TSign runs as follows:

   Case 1: $i = r \notin V$, $j \notin V$, $i \neq j$
   (2.1) $V \leftarrow V \cup \{i, j\}$
   (2.2) $L(i) \leftarrow \mathcal{G}$; $\Sigma(i) \leftarrow \mathsf{SSign}(ssk, i \| L(i))$; $\Delta = \emptyset$
   (2.3) $\delta_{ij} \xleftarrow{R} \mathcal{P}$; $L(j) \leftarrow L(i)^{\delta_{ij}} (mod\ n)$; $\Sigma(j) \leftarrow \mathsf{SSign}(ssk, j \| L(j))$; $\Delta \leftarrow \Delta \cup \{\delta_{ij}\}$
   (2.4) $C_i \leftarrow (i, L(i), \Sigma(i)); C_j \leftarrow (j, L(j), \Sigma(j))$
   (2.5) Return (C_i, C_j, δ_{ij}) as the signature of (i, j)

   Case 2: $i \notin V$, $j = r \notin V$, $i \neq j$
   (2.6) $V \leftarrow V \cup \{i, j\}$
   (2.7) $L(j) \leftarrow \mathcal{G}$; $\Sigma(j) \leftarrow \mathsf{SSign}(ssk, j \| L(j))$; $\Delta = \emptyset$

(2.8) $\delta_{ij} \xleftarrow{R} \mathcal{P}$; $L(i) \leftarrow L(j)^{\delta_{ij}^{-1}(mod\ \varphi(n))}(mod\ n)$; $\Sigma(i) \leftarrow \mathsf{SSign}(ssk, i\|L(i))$; $\Delta \leftarrow \Delta \cup \{\delta_{ij}\}$
(2.9) $C_i \leftarrow (i, L(i), \Sigma(i)); C_j \leftarrow (j, L(j), \Sigma(j))$
(2.10) Return (C_i, C_j, δ_{ij}) as the signature of (i, j)

Case 3: $i \in V$, $j \notin V$, $i \neq j$
(2.11) $V \leftarrow V \cup \{j\}$
(2.12) $\delta_{ij} \xleftarrow{R} \mathcal{P} - \Delta$; $L(j) \leftarrow L(i)^{\delta_{ij}}(mod\ n)$; $\Sigma(j) \leftarrow \mathsf{SSign}(ssk, j\|L(j))$; $\Delta \leftarrow \Delta \cup \{\delta_{ij}\}$
(2.13) $C_j \leftarrow (j, L(j), \Sigma(j))$
(2.14) Return (C_i, C_j, δ_{ij}) as the signature of (i, j)

Case 4: $i \notin V$, $j \in V$, $i \neq j$
(2.15) $V \leftarrow V \cup \{i\}$
(2.16) $\delta_{ij} \xleftarrow{R} \mathcal{P} - \Delta$; $L(i) \leftarrow L(j)^{\delta_{ij}^{-1}}(mod\ n)$; $\Sigma(i) \leftarrow \mathsf{SSign}(ssk, i\|L(i))$; $\Delta \leftarrow \Delta \cup \{\delta_{ij}\}$
(2.17) $C_i \leftarrow (i, L(i), \Sigma(i))$
(2.18) Return (C_i, C_j, δ_{ij}) as the signature of (i, j)

Case 5: $i \notin V$, $j \notin V$, $i, j \neq r$, $i \neq j$. Because $G^* = (V, E^*)$ is a directed tree, there exists an unique undirected path $(r, \alpha_1, \cdots, \alpha_m, i)$ between nodes r and i. Recursively applying algorithm in *Cases 1-4* to each directed edge on the path if its signature does not exist, the signature on (α_m, i) or (i, α_m) can be generated at last. Since there is an unique undirected path between i and j, so $j \neq \alpha_1, \cdots, \alpha_{m-1}$. If $j = \alpha_m$, the signature on (i, α_m) has been already produced. If $j \neq \alpha_m$, apply algorithm in *Case 3* on (i, j). Finally, return (C_i, C_j, δ_{ij}) as the signature of (i, j).

An example for Case 5 is illustrated in Fig. 3, in which there is an undirected path $(r, \alpha_1, \alpha_2, \alpha_3, i)$ between nodes r and i. In order to produce the signature on edge (i, j), signatures on edges (r, α_1) (Case 1), (α_1, α_2) (Case 3), and (α_3, α_2) (Case 4) are firstly generated if they do not exist. Then the signature on edge $(i, \alpha_3(j))$ (Case 4) is produced at last.

Case 6: $i = j$ or $\{i, j\} \in V$, return failure.

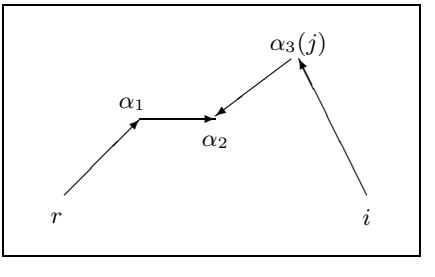

Fig. 3. An example of the signing algorithm TSign in Case 5

3. The deterministic verification algorithm TVf, on inputs $tpk = (n, spk)$, (i, j), and a candidate signature σ_{ij}, proceeds as follows.
(3.1) Parse σ_{ij} as (C_i, C_j, δ_{ij}), parse C_i as $(i, L(i), \Sigma(i))$, parse C_j as $(j, L(j), \Sigma(j))$.
(3.2) If $[(SVf(spk, C_i) = 0) \vee (SVf(spk, C_j) = 0)]$ then return 0.
(3.3) Else if

$$L(i)^{\delta_{ij}} = L(j) \ (mod\ n) \tag{2}$$

then return 1 else return 0.
4. The deterministic composition algorithm $Comp$ takes (i, j, k), signatures σ_{ij} and σ_{jk}, as inputs, and computes a composed signature for the directed edge (i, k) as follows:
(4.1) Parse σ_{ij} as (C_i, C_j, δ_{ij}), parse C_i as $(i, L(i), \Sigma(i))$, parse C_j as $(j, L(j), \Sigma(j))$.
(4.2) Parse σ_{jk} as (C_j, C_k, δ_{jk}), parse C_j as $(j, L(j), \Sigma(j))$, parse C_k as $(k, L(k), \Sigma(k))$.
(4.3) $\delta_{ik} \longleftarrow \delta_{ij} \cdot \delta_{jk}$
(4.4) Return (C_i, C_k, δ_{ik}) as the signature for (i, k).

Proposition 3.1. The $\mathcal{RSADTS}$ directed transitive signature scheme satisfies the correctness requirement of Definition 2.1.

Proof. If (V, L, Δ, Σ) is the internal state of TSign algorithm in $\mathcal{RSADTS}$ scheme, then at any time during the experiment in Fig. 1, the invariant

$$(Legit = false) \vee (\forall (i, j, \sigma_{ij}) \in S, \mathsf{TVf}(i, j, \sigma_{ij}) = 1) \tag{3}$$

holds true.

The above claim is proved by induction on the number of TSign oracle query q as follows.

In the initial state, $S = \emptyset$ and the claim is trivial.

Suppose that the claim is true after $q - 1$ oracle queries.

If $Legit = false$ before the q-th query, it will still be $false$ after the q-th queries. This proves the claim directly.

If the q-th query is a TSign query on (i, j) with $i = j$ or $\{i, j\} \in V$, $Legit$ is set to $false$ and thus the claim is proved. Otherwise, a new element (i, j, σ_{ij}) is added to S, where $\sigma_{ij} = \mathsf{TSign}(tsk, i, j)$. All elements of S satisfying $\mathsf{TVf}(i, j, \sigma_{ij}) = 1$ in the previous state of TSign, still do so in the new state, because TSign only adds new entries to V, L, Δ, Σ, but never change existing entries. Thus, it suffices to show the newly added element (i, j, σ_{ij}) satisfying $\mathsf{TVf}(i, j, \sigma_{ij}) = 1$. This can be seen from the TSign algorithm.

If the q-th query is a Comp query on $(i, j, k), \sigma_{ij}, \sigma_{jk}$ with not all distinct i, j, k, or $(i, j, \sigma_{ij}) \notin S$, or $(j, k, \sigma_{jk}) \notin S$, $Legit$ is set to $false$ and thus the claim is proved. Otherwise, a composed element (i, k, σ_{ik}), where $\sigma_{ik} = (C_i, C_k, \delta_{ik})$ and $\delta_{ik} = \delta_{ij}\delta_{jk}$, is added to S. Because the internal state of the TSign is not affected by the Comp, all elements previously satisfying $\mathsf{TVf}(i, j, \sigma_{ij}) = 1$ will still do so. We only need to verify whether the newly added element (i, k, σ_{ik}) also satisfies

$\mathsf{TVf}(i, j, \sigma_{ij}) = 1$. Since i, j, k are all distinct, $(i, j, \sigma_{ij}) \in S$, and $(j, k, \sigma_{jk}) \in S$, we have $SVf(spk, C_i) = 1$, $SVf(spk, C_k) = 1$, $L(i)^{\delta_{ij}} = L(j)(mod\ n)$ and $L(j)^{\delta_{jk}} = L(k)(mod\ n)$. Furthermore,

$$L(i)^{\delta_{ik}} = L(i)^{\delta_{ij}\delta_{jk}} = (L(i)^{\delta_{ij}})^{\delta_{jk}} = L(j)^{\delta_{jk}} = L(k)\ (mod\ n)$$

Therefore, $TVf(i, k, \sigma_{ik}) = 1$.

A corollary of the claim is that any time during the experiment, $TVf(i, j, \sigma_{ij}) = 1$ for all $(i, j, \sigma_{ij}) \in S$ if $Legit = true$. By this corollary, the verification of a signature in S always succeeds as long as $Legit = true$. Since the experiment outputs $(Legit \wedge NotOK)$ at the end of execution, the claim implies that it returns $false$ for every adversary A, thereby proving this proposition. □

Remark 3.2. Let $n = pq$ where p, q are safe primes, $p' = (p-1)/2$, $q' = (q-1)/2$, $\mathcal{G}$ is an integer such that $\mathcal{G}^2 \neq 1(mod\ n)$, and $g = |<\mathcal{G}>|$, then $p' \mid g$ or $q' \mid g$. When both p' and q' are large, $|<\mathcal{G}>|$ is large, too.

Proposition 3.3. If (V, L, Δ, Σ) is the internal state of TSign algorithm in $\mathcal{RSADTS}$ scheme, then for any $i \neq j$ and $L(i), L(j) \in L$, there exists distinct odd primes $\alpha_1, \cdots, \alpha_\mu, \beta_1, \cdots, \beta_\nu$ in Δ such that

$$L(j) = L(i)^{\alpha_0^{-1}\alpha_1^{-1}\cdots\alpha_\mu^{-1}\beta_0\beta_1\cdots\beta_\nu (mod\ \varphi(n))}\ (mod\ n) \tag{4}$$

where $\alpha_0 = \beta_0 = 1$, $\mu \geq 0$ and $\nu \geq 0$. In addition, $L(j)^2 \neq 1(mod\ n)$ for any $L(j) \in L$.

Proof. Assume that the unique undirected path from i to j contains $\mu + \nu$ edges. There are μ directed edges in the reverse direction from i to j, whose public edge labels are $\alpha_1, \cdots, \alpha_\mu$, while there are ν directed edges in the same direction from i to j, whose public edge labels are $\beta_1, \cdots, \beta_\nu$. Based on TSign algorithm, $\alpha_1, \cdots, \alpha_\mu, \beta_1, \cdots, \beta_\nu$ are distinct primes, and (4) holds.

When $i = r$ is the reference node, we have

$$L(j) = \mathcal{G}^{\alpha_0^{-1}\alpha_1^{-1}\cdots\alpha_\mu^{-1}\beta_0\beta_1\cdots\beta_\nu (mod\ \varphi(n))}\ (mod\ n) \tag{5}$$

If $L(j)^2 = 1(mod\ n)$, then $\mathcal{G}^{2\alpha_0^{-1}\alpha_1^{-1}\cdots\alpha_\mu^{-1}\beta_0\beta_1\cdots\beta_\nu} = 1(mod\ n)$ and thus $\mathcal{G}^2 = 1(mod\ n)$. This contradicts with the assumption that $\mathcal{G}^2 \neq 1(mod\ n)$. Therefore, $L(j)^2 \neq 1(mod\ n)$ for any $L(j) \in L$. □

4 Security Proof

In this section, we prove that $\mathcal{RSADTS}$ scheme is transitively unforgeable under adaptive chosen-message attack if the RSA inversion problem over a cyclic group is hard for the associated generator and the associated standard signature scheme is unforgeable under adaptive chosen-message attack.

Definition 4.1. (RSA Inversion Problem in a Cyclic Group: RSA-icg). Let $k \in \mathcal{N}$ be the security parameter. Let A be an adversary. Consider the experiment $\mathbf{Exp}^{\mathsf{rsa-icg}}_{\mathsf{K}_{\mathsf{rsacg}},\mathsf{A}}(k)$ in Fig. 4.

$(<\mathcal{G}>, n, \varphi(n)) \xleftarrow{R} \mathsf{K}_{\mathsf{rsacg}}(1^k)$, where $n = pq$, p, q are safe primes, $\mathcal{G}^2 \neq 1(mod\ n)$
$e \xleftarrow{R} \mathcal{P}$, $y \xleftarrow{R} <\mathcal{G}>$
$x \leftarrow \mathsf{A}(<\mathcal{G}>, n, e, y)$
If $x^e = y\ (mod\ n)$ then return 1 else return 0

Fig. 4. An experiment to define RSA inversion problem in a cyclic group

The advantage of A is defined as

$$\mathbf{Adv}^{\mathsf{rsa-icg}}_{\mathsf{K}_{\mathsf{rsacg}},\mathsf{A}}(k) = Pr[\mathbf{Exp}^{\mathsf{rsa-icg}}_{\mathsf{K}_{\mathsf{rsacg}},\mathsf{A}}(k) = 1] \tag{6}$$

The RSA-icg problem associated to $\mathsf{K}_{\mathsf{rsacg}}$ is said to be *hard* if the function $\mathbf{Adv}^{\mathsf{rsa-icg}}_{\mathsf{K}_{\mathsf{rsacg}},\mathsf{A}}(k)$ is negligible for any adversary A whose time-complexity is polynomial in the security parameter k.

Remark 4.2. The group $<\mathcal{G}>$ is closed to RSA-icg problem because $x^e = y(mod\ n)$(where the probability of $gcd(e, \varphi(n)) \neq 1$ is negligible) has an unique solution, i.e., $x = y^{e^{-1}}(mod\ n)$, which belongs to $<\mathcal{G}>$.

Remark 4.3. If RSA-icg problem is not hard, the RSA inversion problem is not hard in the case: given (e, y), determine x such that $x^e = y(mod\ n)$, where $n = pq$, p, q are two safe primes, e is an odd prime, and y belongs to $<\mathcal{G}>$. Thus, the hardness of RSA-icg problem is based on the one-wayness of the standard RSA.

Theorem 4.4. Let $\mathsf{K}_{\mathsf{rsacg}}$ be a RSA-based cyclic group generator and $\mathcal{SDS} = (\mathsf{SKG}, \mathsf{SSign}, \mathsf{SVf})$ be a standard digital signature scheme. Let $\mathcal{RSADTS} = (\mathsf{TKG}, \mathsf{TSign}, \mathsf{TVf}, \mathsf{Comp})$ be the directed transitive signature scheme associated to $\mathsf{K}_{\mathsf{rsacg}}$ and $\mathcal{SDS}$. If the RSA-icg problem associated to $\mathsf{K}_{\mathsf{rsacg}}$ is hard and $\mathcal{SDS}$ is unforgeable under adaptive chosen-message attack, then $\mathcal{RSADTS}$ is transitively unforgeable under adaptive chosen-message attack.

Proof. Suppose that we are given a polynomial-time adversary F for $\mathcal{RSADTS}$. It has access to an oracle $\mathsf{TSign}(tsk, \cdot, \cdot)$, by which it is able to obtain a set of transitive signatures, denoted as $S = \{(i, j, \sigma_{ij})\}$. On input tpk and S, F outputs a forgery,

$$\sigma'_{i'j'} = ((i', L_{i'}, \Sigma_{i'}), (j', L_{j'}, \Sigma_{j'}), \delta'_{i'j'}) \tag{7}$$

Let $G = (V, E)$ be the directed graph defined by the set of F's signature queries, where $E = \{(i, j) | \exists (i, j, \sigma_{ij}) \in S\}$ and $V = \{i | (\exists (i, j) \in E) \vee (\exists (j, i) \in E)\}$. Let $\tilde{G} = (V, \tilde{E})$ be the transitive closure of G and $\tilde{S} = \{(i, j, \delta_{ij}) | ((i, j) \in \tilde{E}) \wedge (\mathsf{TVf}(i, j, \delta_{ij}) = 1)\}$. F wins if $(i', j', \sigma'_{i'j'}) \notin \tilde{S}$ and $\mathsf{TVf}(i', j', \sigma'_{i'j'}) = 1$.

There are two cases where F wins as follows:

Case A. In the case where a F's forgery contains recycled node certificates, $L_{i'} = L(i')$ and $L_{j'} = L(j')$, but $(i', j') \notin \tilde{E}$. There exists an unique undirected

path from i' to j' in the transitive reduction of G. After joining i' and j' directly, one cycle containing (i', j') forms. In this cycle, there are two undirected paths from i' to j'. Based on Proposition 3.3, there exists distinct odd primes $\alpha_1, \cdots, \alpha_\mu, \beta_1, \cdots, \beta_\nu$ such that

$$L(i')^{\alpha_0^{-1}\alpha_1^{-1}\cdots\alpha_\mu^{-1}\beta_0\beta_1\cdots\beta_\nu} = L(j') \ (mod\ n) \tag{8}$$

where $\alpha_0 = \beta_0 = 1$ and $\mu, \nu \geq 0$.

In addition, $L(i')^{\delta'_{i'j'}} = L(j')(mod\ n)$. Therefore,

$$L(i')^{\alpha_0\alpha_1\cdots\alpha_\mu\delta'_{i'j'}} = L(i')^{\beta_0\beta_1\cdots\beta_\nu} \tag{9}$$

If $\mu = 0$, the forgery belongs to S because there is a directed path from i to j in G^* and the forgery can be composed. This contradicts with the assumption. Therefore, $\mu \geq 1$.

Next, we can construct a polynomial-time adversary A, to solve the RSA inversion problem in the cyclic group $<\mathcal{G}>$, i.e., determining x such that $x^e = y(mod\ n)$ for given an odd prime e and y in $<\mathcal{G}>$. A proceeds as follows:

First of all, A runs F to obtain $\alpha_1, \cdots, \alpha_\mu, \beta_1, \cdots, \beta_\nu$ and $\delta'_{i'j'}$ such that (9) holds. It is obvious that $\alpha_1 \cdots \alpha_\mu \delta'_{i'j'} \neq \beta_0 \cdots \beta_\nu$. Let $\rho = |\alpha_1 \cdots \alpha_\mu \delta'_{i'j'} - \beta_0 \cdots \beta_\nu|$, then $L(i')^\rho = 1(mod\ n)$. Based on (4), $\mathcal{G}^\rho = 1(mod\ n)$.

If $gcd(e, \rho) = 1$, there are integers s, t such that $se + t\rho = 1$. Therefore, $x = x^{se+t\rho} = y^s(x^\rho)^t(mod\ n)$. Because the probability of $gcd(e, \varphi(n)) \neq 1$ is negligible and y belongs to $<\mathcal{G}>$, so $x = y^{e^{-1}}(mod\ n)$ belongs to $<\mathcal{G}>$. At last, $x^\rho = 1(mod\ n)$ and $x = y^s(mod\ n)$.

If $gcd(e, \rho) \neq 1$ and λ is the largest integer such that $e^\lambda | \rho$, then there are two integers s, t such that $se + t\frac{\rho}{e^\lambda} = 1$ and $x = x^{se+t\frac{\rho}{e^\lambda}} = y^s(x^{\frac{\rho}{e^\lambda}})^t(mod\ n)$. Let $gcd(\rho, \varphi(n)) = \tau$, then $\mathcal{G}^\tau = 1(mod\ n)$. Because the probability of $gcd(e, \varphi(n)) \neq 1$ is negligible, $gcd(e, \tau) = 1$ and $\tau | (\rho / e^\lambda)$. Furthermore, x belongs to $<\mathcal{G}>$ and thus $x^{\frac{\rho}{e^\lambda}} = 1(mod\ n)$. At last, $x = y^s(mod\ n)$.

Case B. In the case where a F's forgery contains at least one node certificate, which includes a signature on a new message, we can construct a polynomial-time adversary B, which is able successfully to make a chosen-message attack to the standard digital signature ($\mathcal{SDS}$) scheme.

Let $\mathbf{E}$ be the event that F's forgery contains recycled node certificates. In case $\overline{\mathbf{E}}$ happens, A aborts. In case $\mathbf{E}$ happens, B gives up. Accordingly, we have

$$\begin{aligned}
&\mathbf{Adv}^{\mathsf{dtu-cma}}_{\mathcal{RSADTS},\mathsf{F}}(k) = Pr[\mathbf{Exp}^{\mathsf{dtu-cma}}_{\mathcal{RSADTS},\mathsf{F}}(k) = 1] \\
&= Pr[\mathbf{Exp}^{\mathsf{dtu-cma}}_{\mathcal{RSADTS},\mathsf{F}}(k) = 1 \wedge \mathbf{E}] + Pr[\mathbf{Exp}^{\mathsf{dtu-cma}}_{\mathcal{RSADTS},\mathsf{F}}(k) = 1 \wedge \overline{\mathbf{E}}] \\
&\leq \mathbf{Adv}^{\mathsf{rsa-icg}}_{\mathsf{K_{rsacg}},\mathsf{A}}(k) + \mathbf{Adv}^{\mathsf{uf-cma}}_{\mathcal{SDS},\mathsf{B}}(k)
\end{aligned}$$

If the RSA-icg problem associated to $\mathsf{K_{rsacg}}$ is hard and $\mathcal{SDS}$ is unforgeable under adaptive chosen-message attack, then $\mathcal{RSADTS}$ is transitively unforgeable under chosen-message attack.

The theorem is proved. □

5 Performance Analysis

In this section, we analyze performance of $\mathcal{RSADTS}$ scheme in terms of signature size, computation cost and compare $\mathcal{RSADTS}$ with those undirected transitive signature schemes using node certificates.

At first, let us consider the size of a transitive signature (C_i, C_j, δ_{ij}) on an edge (i, j), where $C_i = (i, L(i), \Sigma(i))$, $C_j = (j, L(j), \Sigma(j))$. The size of $L(i)$ and $L(j)$ amounts to $2\log_2 n$, $\Sigma(i)$ and $\Sigma(j)$ are two standard signatures, and i, j are integers. Therefore, the size of C_i or C_j is fixed. However, the size of edge label δ_{ij}, which is either a prime or a product of some primes (in the composition case), varies case by case. In practice, we can choose small primes as δ_{ij} for edges in the transitive reduction of a directed graph so as to reduce the size of their possible compositions. This will not affect security of $\mathcal{RSADTS}$ scheme.

Even though the distribution of primes seems random, the number of primes less than an integer is surprisingly well behaved. Let $p(\lambda)$ be the λ-th prime, it has been shown in [19][20] that $p(\lambda) \sim \lambda\ ln(\lambda)$.

Let $|V|$ be the number of nodes in a directed graph, then the transitive reduction has $|V| - 1$ edges, which need $|V| - 1$ distinct primes for edge labels. In addition, Let m be the number of directed edges on the longest directed path.

If we assign the first $|V| - 1$ odd primes to $|V| - 1$ edges in E^*, the average size of a single edge label is about $\log_2(|V|ln(|V|))/2$ bits. For a signature which is composed by the longest directed path, the average size of the composed edge label is about $M = m\log_2(|V|ln(|V|))/2$ bits. Some examples are given in Table 1.

Table 1. The size of edge label δ_{ij} $(|\delta_{ij}| = \log_2 \delta_{ij})$

$(\|V\|, m)$	$\|\delta_{ij}\| : (i,j) \in E^*$	$M = max\{\|\delta_{ij}\|\}$
$(100, 10)$	≈ 4.5 bits	≈ 45 bits
$(500, 50)$	≈ 6 bits	≈ 300 bits
$(1000, 100)$	≈ 6.5 bits	≈ 650 bits
$(10000, 100)$	≈ 8 bits	≈ 800 bits
$(100000, 100)$	≈ 10 bits	≈ 1000 bits

Next, let us consider the computation cost of $\mathcal{RSADTS}$ scheme. In order to generate a transitive signature for (i, j), two node certificates (C_i, C_j) are needed to compute, which involves one modular exponentiations for computing $L(j)$ or $L(i)$ and two standard signatures. The verification of a transitive signature requires to check that two node certificates and $L(i)^{\delta_{ij}} = L(j)(mod\ n)$, involving the verification of two standard signatures and the computation of one modular exponentiation. The composition algorithm is efficient, involving only one integer multiplication.

A directed transitive scheme can be trivially realized by accepting, as a valid signature of $\{i, j\}$, any chain of signatures that authenticates a sequence of edges forming a path from i to j. Two issues lead to exclude this trivial solution: the

growth in signature size, and the loss of privacy incurred by having signatures carry information about their history [1].

In $\mathcal{RSADTS}$ scheme, the verification of a composed transitive signature do not require information of intermediary nodes. Therefore, privacy of a directed graph can be kept. Although the signature size of a composed transitive signature in $\mathcal{RSADTS}$ scheme does increase with the growth of the edges, the growth rate is much slower than the trivial solution.

For example, suppose that a directed graph has about 10000 nodes, each time when a new node is added, which results in a new directed edge, the size of a composed signature in $\mathcal{RSADTS}$ scheme increases about 8 bits in average. However, the size of a composed signature in the trivial solution increases about 1024 bits if RSA signature scheme is used (where the RSA modulus has 1024 bits). The growth rate of the composed signature size in the trivial solution is almost 128 times of that in $\mathcal{RSADTS}$ scheme.

Performance comparison of $\mathcal{RSADTS}$ with those undirected transitive signature schemes using node certificates is shown in Table 2.

Table 2. Performance comparison among transitive signature schemes ($|n| = \log_2 n$)

Scheme	Signing cost	Verification cost	Composition cost	Signature size
$\mathcal{DLTS}$	2 stand. signs. 2 exp. in $\mathbf{G}$	2 stand. verifs 1 exp. in $\mathbf{G}$	2 adds in $\mathbf{Z_q}$	2 stand. signs. 2 points in $\mathbf{G}$ 2 points in $\mathbf{Z_q}$
$\mathcal{RSATS}$-1	2 stand. signs. 2 RSA encs	2 stand. verifs 1 RSA enc.	$O(\|n\|^2)$ ops	2 stand. signs. 3 points in $\mathbf{Z_n^*}$
$\mathcal{FactTS}$-1	2 stand. signs. $O(\|n\|^2)$ ops	2 stand. verifs $O(\|n\|^2)$ ops	$O(\|n\|^2)$ ops	2 stand. signs. 3 points in $\mathbf{Z_n^*}$
$\mathcal{DLTS}$-1$\mathcal{M}$	2 stand. signs. 2 exp. in $\mathbf{G}$	2 stand. verifs 1 exp. in $\mathbf{G}$	1 add in $\mathbf{Z_q}$	2 stand. signs. 2 points in $\mathbf{G}$ 1 points in $\mathbf{Z_q}$
$\mathcal{GapTS}$-1	2 stand. signs. 2 exp. in $\hat{\mathbf{G}}$	2 stand. verifs 1 $\mathsf{S_{ddh}}$	$O(\|n\|^2)$ ops	2 stand. signs. 3 points in $\mathbf{G}$
$\mathcal{RSADTS}$	2 stand. signs. 1 exp. in $<\mathcal{G}>$	2 stand. verifs 1 exp. in $<\mathcal{G}>$	$\leq \|M\|$ ops	2 stand. signs. 2 points in $<\mathcal{G}>$ 1 label $\delta_{ij} \leq M$

In Table 2, the word "stand." refers to operations of the underlying standard signature scheme, $\mathbf{G}$ denotes the group of prime order q, and n denotes a product of two primes, $\hat{\mathbf{G}}$ is a gap Diffie-Hellman group and $\mathsf{S_{ddh}}$ refers to the decision Diffie-Hellman algorithm in $\hat{\mathbf{G}}$. Abbreviations used are: "exp." for an exponentiation in the group; "RSA enc." for an RSA encryption; "RSA dec." for an RSA decryption; and "ops." for the number of elementary bit operations.

From Table 2, we can see that $\mathcal{RSADTS}$ scheme has almost the same signing and verification costs as other undirected transitive signature schemes (excluding

$\mathcal{F}act\mathcal{TS}$-1). But its composition cost and signature size vary according to the number of nodes $|V|$ in a directed graph and the number of directed edges m on the longest directed path. When $M \leq n$, $\mathcal{RSADTS}$ scheme has even better performance than $\mathcal{RSATS}$-1.

In practice, directed paths of a directed graph are not very long. For example, in a directed graph for a military chain-of-command, the longest directed path usually contains less than 100 edges. In this case, it can be seen from Table 1 that $\mathcal{RSADTS}$ scheme is practical and efficient.

$\mathcal{RSADTS}$ scheme allows dynamically to add a new node, which results in a new directed edge, into a directed graph. In other word, the directed graph can dynamically grow. However, it does not allow to create a new edge (i, j) by connecting two existing nodes i and j.

$\mathcal{RSADTS}$ scheme can be applied to a directed graph whose transitive reduction is a disjoint union of directed trees, where transitive signatures in different directed trees are distinguished with different tree labels.

6 Conclusion

In 2002, Micali and Rivest raised an open problem as to whether directed transitive signatures exist or not. In this paper, we have proposed a natural RSA based directed transitive signature scheme $\mathcal{RSADTS}$ for a directed graph whose transitive reduction is a directed tree. $\mathcal{RSADTS}$ scheme has been proved to be transitively unforgeable under adaptive chosen-message attack if the RSA inversion problem over a cyclic group is hard and the underlying standard signature scheme is unforgeable under adaptive chosen-message attack. Therefore, we have answered the open problem in the case where the transitive reduction of a directed graph is a directed tree. Furthermore, performance analysis has shown that $\mathcal{RSADTS}$ scheme is practical and efficient. When $M \leq n$, $\mathcal{RSADTS}$ scheme has even better performance than $\mathcal{RSATS}$-1.

References

1. S. Micali and R. Rivest, "Transitive signature schemes", *Proc. CT-RSA'02*, pp. 236-243, San Jose, CA, USA, Feb. 2002.
2. M. Bellare and G. Neven, "Transitive signature based on factoring and RSA", *Proc. Asiacrypt'02*, pp. 397-414, Queenstown, New Zealand, Dec. 2002.
3. M. Bellare and G. Neven, "Transitive signatures: new schemes and proofs", *IEEE Transactions on Information Theory*, vol. 51, no. 6, pp. 2133-2151, 2005.
4. T. Okamoto, "Provably secure and practical identification schemes and corresponding signature schemes", *Proc. Crypto'92*, pp. 31-53, 1993.
5. C. P. Schnorr, "Efficient identification and signatures for smart cards", *Proc. Crypto'89*, pp. 239-252, 1989.
6. S. F. Shahandashti, M. Salmasizadeh, and J. Mohajeri, "A provably secure short transitive signature scheme from bilinear group pairs", *Proc. SCN'04*, pp. 60-76, Amalfi, Italy, Sept 2004.

7. S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks", *SIAM Journal of Computing*, vol. 17, no. 2, pp. 281-308, 1988.
8. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, "The One-more-RSA-inversion problems and the security of Chaum's blind signature scheme", *Journal of Cryptology*, vol. 16, no. 3, pp. 185-215, 2003.
9. D. Chaum, "Blind signatures for untraceable payments", *Proc. Crypto'82*, pp. 199-203, 1982.
10. M. Bellare and A. Palacio, "GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attack", *Proc. Crypto'02*, pp. 162-177, Aug 2002.
11. L. C. Guillou and J. J. Quisquater, "A 'paradoxical' identity-based signature scheme resulting from zero-knowledge", *Proc. Crypto'88*, pp. 216-231, 1988.
12. D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing", *Proc. Asiacrypt'01*, pp. 514-532, 2001.
13. A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme", *Proc. Public-Key Cryptography 2003*, pp. 31-46, 2003.
14. M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols", *Proc. 1st Conf. Computer and Communications Security*, pp. 62-73, Fairfax, VA, Nov 1993.
15. S. R. Hohenberger, "The cryptographic impact of groups with infeasible inversion", Master's Thesis, MIT, MA, May 2003.
16. A. R. Sadeghi and M. Steiner, "Assumptions related to discrete logarithms: Why subtleties make a real difference", *Proc. Eurocrypt'01*, pp. 244-261, 2001.
17. H. Kuwakado and H. Tanaka, "Transitive signature scheme for directed trees", *IEICE Trans. Fundamentals*, vol.E86-A, no. 5, pp. 1120-1126, May 2003.
18. X. Yi, C. H. Tan and E. Okamoto, "Security of Kuwakado-Tanaka transitive signature scheme for directed trees", *IEICE Trans. Fundamentals*, vol.E87-A, no. 4, pp. 955-957, Apr 2004.
19. G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 1979.
20. P. Ribenboim, *The New Book of Prime Number Records*, 3rd Edition, Springer-Verlag, New York, NY, 1995.

Identity-Based Multi-signatures from RSA

Mihir Bellare[1] and Gregory Neven[2]

[1] Department of Computer Science & Engineering
University of California San Diego
mihir@cs.ucsd.edu
http://www.cs.ucsd.edu/users/mihir
[2] Department of Electrical Engineering
Katholieke Universiteit Leuven
and Département d'Informatique
Ecole Normale Supérieure
Gregory.Neven@esat.kuleuven.be
http://www.neven.org

Abstract. Multi-signatures allow multiple signers to jointly authenticate a message using a single compact signature. Many applications however require the public keys of the signers to be sent along with the signature, partly defeating the effect of the compact signature. Since identity strings are likely to be much shorter than randomly generated public keys, the identity-based paradigm is particularly appealing for the case of multi-signatures. In this paper, we present and prove secure an identity-based multi-signature (IBMS) scheme based on RSA, which in particular does not rely on (the rather new and untested) assumptions related to bilinear maps. We define an appropriate security notion for interactive IBMS schemes and prove the security of our scheme under the one-wayness of RSA in the random oracle model.

1 Introduction

With the increased adoption of small, energy-restricted devices such as laptops, cell phones, PDAs and sensors, battery life has become a crucial bottleneck in the usage of these devices — and an important distinguishing factor in their sales. Fast progress is being made in the development of lighter and higher-capacity batteries, but at the same time the demand for energy-preserving technology is more pressing than ever. Much effort is being put in the design of low-power microprocessors, but also the software running on these processors is being optimized for energy consumption, rather than for speed or portability.

In accordance with their wireless nature, communication on these portable devices often takes place over wireless channels such as Bluetooth and WiFi. Unfortunately, these communication mechanisms are rather expensive in terms of energy consumption. Reducing the number of bits to communicate is crucial to increase battery life: communicating a single bit of data requires significantly more power than executing a 32-bit instruction [1], so it makes perfect sense to invest extra computation cycles to save on bandwidth. Also, communication

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 145–162, 2007.

is often not reliable, so the fewer the number of bits one has to communicate, the better. To make things worse, wireless channels are inherently vulnerable to eavesdropping and tampering attacks by outsiders. Strong cryptography is needed to protect the communication, adding even more overhead to the communication. It is our challenge as designers of cryptographic primitives to limit this overhead to a minimum.

Multi-signature schemes. A multi-signature (MS) scheme [20] allows n different signers with public keys $pk_1, \ldots, pk_n$ to collectively sign a message m, yielding a multi-signature σ of roughly the same size as a standard signature, yet that certifies m under all public keys $pk_1, \ldots, pk_n$ simultaneously. By transmitting σ instead of n individual signatures, multi-signature schemes can help greatly to save on communication costs.

However, one still needs the public keys of all cosigners in order to verify the validity of such a multi-signature. In most applications these public keys will have to be transmitted along with the multi-signature, which partially defeats the primary purpose of using a multi-signature scheme, namely to save on bandwidth. The inclusion of some information that uniquely identifies the cosigners seems inevitable for verification, but often this information can be represented more succinctly than by means of randomly generated public keys. For example, the signers' user names or IP addresses could suffice for this purpose; this information may even already be present in package headers. Moreover, each public key may come with an associated certificate containing a signature from a certification authority (CA) and the CA's public key, which on its turn may come with a chain of certificates leading to the root CA. Altogether, this sums up to many more bits being transmitted than strictly necessary to authenticate the message.

Identity-based signatures. In an identity-based signature scheme [28], the public key of a user is simply his identity, e.g. his name, email or IP address. A trusted key distribution center provides each signer with the secret signing key corresponding to his identity. When all signers have their secret keys issued by the same key distribution center, individual public keys become obsolete, removing the need for explicit certification and all associated costs. These features make the identity-based paradigm particularly appealing for use in conjunction with multi-signatures, leading to the concept of identity-based multi-signature (IBMS) schemes.

Generic constructions. In spite of their appeal with regard to applications, implementations of IBMS schemes are rather limited. As demonstrated in [12,2], any standard signature scheme can be transformed into an identity-based one using the "certification paradigm". One can attempt to derive IBMS schemes from existing standard MS schemes via this approach [16]. The problem is that the resulting multi-signature is not compact due to the need to include the certificates with each signature. Even if the signatures in the certificates can be aggregated [6], the public keys they contain cannot. In summary, unlike the case of standard signatures, there seems no trivial, general way to transform compact signature schemes into identity-based ones.

An existing construction. The only provably secure IBMS scheme known today is due to Gentry and Ramzan [17]. The scheme employs groups with bilinear maps (also known as pairings), which are usually implemented by modified Weil or Tate pairing over elliptic or hyperelliptic curves. To avoid putting all our eggs in the same basket, it is common practice in cryptography to try to find alternative constructions of a primitive based on different assumptions. While pairings have turned out extremely useful in the design of cryptographic protocols, they were only recently brought to the attention of cryptographers [21], and hence did not yet enjoy the same exposure to cryptanalytic attacks by experts as other, older problems from number theory such as discrete logarithms, factoring and RSA. This exposure is necessary to build confidence in the hardness of the underlying problems; without it, their use in high-security applications may not be advisable.

Also, efficient implementations of RSA are ubiquitous, even in the public domain, while implementations of pairings are much harder to come by. Unlike RSA, even building an inefficient prototype implementation of pairings is far from straightforward for anyone but an expert, and even then it is often difficult or impossible to generate curves with the desired security parameters [15]. Companies may have invested in expensive hardware or software implementations of RSA, and may be reluctant to reinvest in new pairing implementations.

Our contributions. We present an efficient and provably secure IBMS scheme based on RSA, which is thereby the first provably secure IBMS scheme not relying on the use of pairings. Our scheme is essentially a multi-signature variant of the Guilliou-Quisquater (GQ) identity-based signature scheme [18], strengthened with techniques from [3] to provide security against concurrent attacks. Unstrengthened variants of our scheme were proposed before (but without security proofs) in [18,8]. The proof makes use of the general forking lemma of [3], which, unlike the original forking lemma by Pointcheval and Stern [27], applies to more general contexts than generic standard signature schemes. Signatures under our scheme are 1184 bits long for typical values of the security parameters, which is longer than the 320-bit signatures of the scheme of [17]. Verification on the other hand is considerably cheaper: our scheme needs only a single (multi-) exponentiation in an RSA group, as opposed to three pairing computations for the scheme of [17]. The cost of one pairing computation is roughly that of 6–20 exponentiations.

We prove our scheme secure in the random oracle model under the one-wayness of RSA. Unlike the scheme of [17], our scheme requires the signers to interact to generate a signature, so we had to extend their security notion to model this interaction in the presence of an adversary, taking inspiration from the (non-identity-based) notions of [24,3]. We consider the strongest possible setting, namely with insecure and unauthenticated communication links controlled by the adversary, without assuming the availability of a trusted broadcast primitive. In fact, we distinguish between two different notions, called *single-signer* and *multi-signer* security, based on the number of signers whose role can be played by the signing oracle. While not obvious at first, we prove that these

notions are in fact equivalent, so that we can prove our scheme secure under the simpler single-signer notion. As in [3], but unlike [24], we allow the adversary to concurrently engage in as many arbitrarily interleaved signature protocols as it wants.

Interactive vs. non-interactive schemes. As noted above, our scheme requires the signers to interact in order to generate a signature. The IBMS scheme of [17] is non-interactive, meaning that each signer independently computes its share to the signature, and anyone can combine these shares into a compact signature. The requirement of interaction may seem to conflict with our goal of saving on bandwidth. We argue that this is not always the case. Consider for example a wired network of sensors in a very remote location (e.g. in a desert, or in space) that needs to report back to a far-away base station through wireless communication. The sensors can use the cheap wired network to execute joint signing protocols and send the resulting compact signatures over the expensive wireless channel. A non-interactive scheme does not offer any real advantage here. In particular, it does not remove the need for local communication: the sensors still need a round of interaction to exchange signature shares. In general, the added cost of interaction depends highly on the network topology.

Aggregate vs. multi-signatures. Aggregate signature (AS) schemes [6] can be seen as a generalization of multi-signatures where each signer i signs a different message m_i. Only a single identity-based aggregate signature (IBAS) scheme is known [17]; it is also based on pairings. IBAS schemes automatically give rise to IBMS schemes, but the scheme resulting from the only known IBAS instantiation [17] is less efficient than their direct IBMS construction. We note that the distinction between aggregate and multi-signatures becomes irrelevant for interactive schemes. Indeed, one can easily construct an interactive aggregate signature scheme from a multi-signature scheme by letting the signers, in a first round of communication, inform each other about the messages m_i they are about to sign. The common message m can then be taken to be the concatenation of (ID_i, m_i) tuples. Hence, the single-message restriction of multi-signature schemes is not really limiting in the case of interactive schemes.

Other related work. Cheng et al. [10] recently proposed another interactive IBMS scheme based on pairings, but proved it secure only under a weak variant of selective-ID security. To the best of our knowledge, the schemes of [10,17] are the only instantiations of IBMS in the literature.

There is more work on compact signature schemes in the non-identity-based setting. There is a vast literature on MS schemes, but the only provably secure schemes are those of [24,5,22,3]. The schemes of [5,22] are based on pairings, those of [24,3] on discrete logarithms. In a sequential aggregate signature (SAS) scheme [23], aggregation cannot be performed by an outsider; instead, the signers cooperate, each in turn aggregating their signature into the current aggregate using their secret key. The only known instantiations of SAS schemes are due to [23,22]. The scheme of [23] is based on families of certified trapdoor permutations, of which strictly speaking no instantiations exist, but the authors discuss

how to instantiate their scheme with RSA. The scheme of [22] uses pairings, and is the only one with security in the standard (i.e., non-random-oracle [4]) model.

2 Identity-Based Multi-signatures

NOTATION. Let $\mathbb{N} = \{1, 2, 3, \ldots\}$. A string means a binary one. The empty string is denoted ε. If x, y are strings, then $|x|$ is the length of x. If $x_1, x_2, \ldots$ are objects then $x_1 \| x_2 \| \ldots$ denotes an encoding of them as strings from which the constituent objects are easily recoverable. If S is a (multi)set, then $|S|$ is its cardinality, $s \xleftarrow{\$} S$ denotes the operation of assigning to s an element of S chosen at random, and $\langle S \rangle$ is a unique encoding of S as a string. If A is a randomized algorithm, then $\mathsf{A}(x_1, \ldots; \rho)$ denotes its output on inputs $x_1, \ldots$ and coins ρ, while $y \xleftarrow{\$} \mathsf{A}(x_1, \ldots)$ means that we choose ρ at random and let $y = \mathsf{A}(x_1, \ldots; \rho)$.

GENERAL SETTING. We adapt definitions from [24,3] to the identity-based setting. Consider n different signers with identities $ID_1, \ldots, ID_n$ who collectively want to sign the same message m so that the resulting signature σ authenticates m under each of their identities. We consider schemes with interactive signing algorithms, meaning that all signers are simultaneously online and interact to produce the signature σ. We assume that signers interact in rounds, where at the beginning of each round each signer receives an incoming message from each of the other signers, performs some computation and sends an outgoing message to all other signers. We let the incoming message of the first round be the local input of the signing algorithm, consisting of the secret key, the list of co-signers, and the message m. The outgoing message of the last round is the final signature σ, or $\perp$ to indicate failure.

We assume that each signer has a direct connection to each of its co-signers. We do not assume these connections to be private or authenticated however, and neither do we assume the availability of an atomic broadcast primitive. When describing signing protocols, we let each signer refer to itself as signer 1 with identity ID_1, and let it assign an index $2, \ldots, n$ to each of its cosigners with identities $ID_2, \ldots, ID_n$. These indices serve merely as a local pointer for the signer to distinguish between its different cosigners and the connections over which it communicates with them. They have no global meaning outside this protocol instance: the signer that you refer to as signer 2 with identity ID_2 may very well be my signer 3 with identity ID_3, and in a later protocol instance I may very well refer to the same signer as signer 4 with identity ID_4. The indices have no certified relationship with identities, and are certainly not included in the identity strings.

SYNTAX OF IBMS SCHEMES. Formally an identity-based multi-signature scheme $\mathsf{IBMS} = (\mathsf{Setup}, \mathsf{KeyDer}, \mathsf{Sign}, \mathsf{Vf})$ consists of four algorithms. A trusted key distribution center runs the Setup algorithm to generate a master public key mpk and corresponding master secret key msk. To a signer with identity $ID \in \{0,1\}^*$, it provides a secret key derived via $sk_{ID} \xleftarrow{\$} \mathsf{KeyDer}(msk, ID)$. The signer can

use this secret key to participate in signing protocols as prescribed by the Sign algorithm, which takes as additional input the message m and a multiset $L = \{ID_1, \ldots, ID_n\}$ containing the identities of all signers participating in the protocol. After a number of interactions, the Sign algorithm outputs the multi-signature σ, or $\perp$ to indicate failure. The verification algorithm Vf takes as input the master public key mpk, a multiset of identities $L = \{ID_1, \ldots, ID_n\}$, the message m and a candidate signature σ, and outputs 1 if σ is a valid signature on message m by all identities in L, or outputs 0 otherwise. (Because a cheating signer may try to impersonate an identity for which he does not have the key, we explicitly allow multiple occurrences of the same identity by modeling L as a multiset.)

In the random oracle model [4], the key derivation, signing and verification algorithms additionally have access to a random oracle $\mathrm{H}(\cdot) : \{0,1\}^* \to D$, where D depends on the scheme and possibly on the master public key mpk. If the scheme uses multiple random oracles, these can be derived from a single oracle H using techniques of [4].

Correctness requires that, whenever all signers correctly follow the Sign protocol using secret key $sk_{ID_i} \xleftarrow{\$} \mathsf{KeyDer}(msk, ID_i)$, then with probability one they all end up with local output a signature σ such that $\mathsf{Vf}(mpk, L, m, \sigma) = 1$ for all positive integers n, all (mpk, msk) output by Setup, all $ID_1, \ldots, ID_n \in \{0,1\}^*$ and all messages $m \in \{0,1\}^*$.

3 Two Security Notions and Their Equivalence

In standard (i.e., non-identity-based) multi-signature schemes, security is commonly defined through an experiment with a single honest "target" signer, effectively viewing all other signers as corrupted [24,5,3]. Security requires that it be infeasible to forge a multi-signature involving the target signer. The adversary has access to a signing oracle that plays the role of the target signer, while the adversary plays the role of all other signers participating in the protocol. The logic underlying this simplified model is that any honestly generated public key is "as good" as any other one; no adversary is expected to perform significantly better in a model with multiple honest signers, as it could easily have simulated these other signers itself.

The same logic does *not* go through for IBMS schemes however. Any identity is not necessarily "as good" as any other one: the scheme may behave differently on different identities, or may even have "weak" identities for which forging signatures is easy. We therefore need to consider a stronger security notion where the adversary can adaptively decide to corrupt signers by submitting their identities to a key derivation oracle, resulting in it being given their secret signing keys.

The adversary is also given access to a signing oracle through which it can engage in any number of arbitrarily interleaved signing protocols with honest signers. Unlike the case of standard multi-signature schemes however, it is not immediately clear whether it is sufficient to let the oracle in each protocol instance play the role of a single honest signer, leaving it up to the adversary to

play the role of all other signers, or whether we should allow the oracle to play the role of multiple honest signers simultaneously. Indeed, an adversary in the former model could try to simulate the signing oracle of the latter model by corrupting all but one of the honest signers, but this precludes attacking any of the corrupted identities in the final forgery. Our goal is of course to achieve the strongest security notion possible, but at the same time we want to avoid making security proofs unnecessarily complicated. Since the relation between the two notions is not immediately clear, we present both in full detail below, and then prove that both notions are in fact equivalent (up to a factor that is the product of the number of the adversary's signature queries and the maximal number of participating signers in one protocol).

Our security notions do not cover *robustness* [7], meaning that we do not prevent malicious signers or network faults from disrupting the signing protocol or from making honest signers end up with invalid signatures. As argued in [24], the strong notion of unforgeability in an adversarially-controlled network strengthens the security guarantees offered by our scheme, but prevents robustness. Indeed, if a signer with identity ID refuses to cooperate or is unreachable due to network faults, then it should be impossible for the other signers to compute any signature involving ID, for otherwise the scheme would be forgeable.

SINGLE-SIGNER SECURITY. In somewhat more detail, we consider the following three-phase game associated to multi-signature scheme $\mathsf{IBMS} = (\mathsf{Setup}, \mathsf{KeyDer}, \mathsf{Sign}, \mathsf{Vf})$ and adversary (forger) F:

Setup: The game generates a master key pair $(mpk, msk) \xleftarrow{\$} \mathsf{Setup}$, and gives the master public key mpk as input to the forger.

Attack: F can decide to corrupt a signer by querying a key derivation oracle with any identity ID, which returns the secret key for that identity $sk_{ID} \xleftarrow{\$} \mathsf{KeyDer}(msk, ID)$. In the random oracle model [4], it also has access to a random oracle $\mathrm{H}(\cdot)$. The forger can engage in an instance of the signature protocol with any honest signer ID that it chooses, while F itself plays the role of all other signers. It does so by submitting the identity $ID \in \{0,1\}^*$, a multiset of identities L and a message m to a signing oracle. The multiset L contains ID at least once. The oracle plays the role of ID as dictated by the Sign algorithm; the role of the other (possibly cheating) signers in L is played by F. Note that the identities in $L \setminus \{ID\}$ need not all be corrupted; the forger is free to try to simulate them without their secret key. The forger can schedule an arbitrary number of protocol instances concurrently, interacting with "clones" of the same honest signer, where each clone maintains its own state and uses its own coins. When the honest signer terminates a signing protocol, its local output (whether $\perp$ or a compact signature σ) is returned to F.

Forgery: At the end of its execution, F outputs a multiset $L = \{ID_1, \ldots, ID_n\}$ of identities, a message m and a forged signature σ. The forger is said to win the game if $\mathsf{Vf}(mpk, L, m, \sigma) = 1$, if L contains at least one uncorrupted identity, and if the forger never submitted a query (ID, L, m) to the signing oracle for any $ID \in L$.

The advantage of F in breaking the single-signer unforgeability of IBMS is defined as the probability that F wins the above game, where the probability is taken over the coin tosses of the forger, the honest signers, and the setup phase. We say that a forger F $(t, q_{\mathrm{K}}, q_{\mathrm{S}}, q_{\mathrm{H}}, n_{\max}, \epsilon)$-breaks the single-signer security of IBMS if it runs in time at most t; performs at most q_{K} key derivation queries and at most q_{H} random oracle queries; engages in at most q_{S} signature interactions with up to $n_{\max}$ signers; and has advantage at least ϵ. (If there is more than one random oracle, q_{H} denotes the sum of the number of queries to all random oracles.) We say that IBMS is $(t, q_{\mathrm{K}}, q_{\mathrm{S}}, q_{\mathrm{H}}, n_{\max}, \epsilon)$ single-signer secure in the random oracle model if no forger F $(t, q_{\mathrm{K}}, q_{\mathrm{S}}, q_{\mathrm{H}}, n_{\max}, \epsilon)$-breaks it.

Multi-signer security. Alternatively, we define the notion of *multi-signer security*. The game is similar to the one described above, except that the signing oracle in each protocol instance can play the role of multiple honest signers simultaneously. In particular, it performs signature queries by submitting two multisets of identities $L_{\mathrm{h}}, L_{\mathrm{c}}$ and a message m to the signing oracle. The multiset L_{h} contains the identities of honest signers, whose role will be played by the oracle as dictated by the Sign algorithm. The forger plays the role of the (possibly) cheating signers contained in L_{c}.

For the communication between signers, we consider the strongest possible notion: the adversary completely controls all network traffic, even between honest signers. We model this by letting honest signers, when they want to send a message to another signer, hand the message to the adversary for delivery. The adversary can then choose to inspect, modify and whether or not to deliver the message at will. We do not assume the availability of a trusted broadcast primitive, so the forger can cause different honest signers to have a different view of the protocol by providing them with different messages. Note that this is a situation that in particular cannot arise in the single-signer notion.

The forger F is said to win the game if eventually F outputs a forgery (L, m, σ) such that $\mathsf{Vf}(mpk, L, m, \sigma) = 1$, L contains at least one uncorrupted identity, and the forger never performed a signature query $(L_{\mathrm{h}}, L_{\mathrm{c}}, m)$ where $L_{\mathrm{h}} \cup L_{\mathrm{c}} = L$. The advantage of F in breaking the multi-signer security of IBMS is defined as the probability that it wins the above game. We say that a forger F $(t, q_{\mathrm{K}}, q_{\mathrm{S}}, q_{\mathrm{H}}, n_{\max}, \epsilon)$-breaks the multi-signer security of IBMS if has advantage at least ϵ and adheres to the resource restrictions as explained above. We say that IBMS is $(t, q_{\mathrm{K}}, q_{\mathrm{S}}, q_{\mathrm{H}}, n_{\max}, \epsilon)$ multi-signer secure if no forger F $(t, q_{\mathrm{K}}, q_{\mathrm{S}}, q_{\mathrm{H}}, n_{\max}, \epsilon)$-breaks it.

Single-signer implies multi-signer security. It is obvious that the notion of multi-signer security is at least as strong as that of single-signer security, since the latter can be viewed as a special case of the former where each of the sets L_{h} is restricted to be a singleton. The following theorem states the less obvious direction that single-signer security also implies multi-signer security, at the loss of a factor $n_{\max} q_{\mathrm{H}}$ in the reduction.

Theorem 1. *If the IBMS scheme* IBMS *is* $(t', q'_{\mathrm{K}}, q_{\mathrm{S}}, q_{\mathrm{H}}, n_{\max}, \epsilon')$ *single-signer secure, then it is also* $(t, q_{\mathrm{K}}, q_{\mathrm{S}}, q_{\mathrm{H}}, n_{\max}, \epsilon)$ *multi-signer secure for*

$\epsilon \geq n_{\max}(q_S + 1) \cdot \epsilon'$, $t \leq t' - n_{\max} q_S \cdot t_{\mathsf{Sign}}$ *and* $q_K \leq q'_K - n_{\max} q_S$, *where* t_{Sign} *is the running time of an execution of the signing protocol.*

Proof. Let F be a forging algorithm that $(t, q_K, q_S, q_H, n_{\max}, \epsilon)$-breaks the multi-signer security of IBMS. Consider the single-signer forging algorithm F' that, on input mpk, guesses indices $i \xleftarrow{\$} \{0, \ldots, q_S\}$ and $j \xleftarrow{\$} \{1, \ldots, n_{\max}\}$ and runs $\mathsf{F}(mpk)$. F' maintains a counter ctr, initially zero, and an identity string ID^*, initially $\perp$. It responds to F's signature queries (L_h, L_c, m) as follows. It first increases ctr; if $ctr = i$, it sets ID^* to the j-th identity in L_h, or aborts if L_h contains less than j elements. If F' already corrupted ID^* before, it also aborts. To simulate the signing protocol, F' uses its own signing oracle on input $(ID^*, L_h \cup L_c, m)$ to simulate ID^*, and corrupts all other identities in L_h so that it can correctly simulate them using their secret keys.

When F queries for the secret key of ID^*, F' gives up; otherwise, it forwards the response from its own key derivation oracle. Eventually, F outputs its forgery (L, m, σ). To be a valid forgery, L must contain at least one identity ID that F never corrupted. If $ID = ID^*$, then (L, m, σ) is also a valid forgery for F'. Likewise, if $i = 0$ and F never performed any signature queries involving ID, then (L, m, σ) is a valid forgery for F'. In all other cases, F' aborts.

It is easy to see that F' succeeds with probability $\epsilon' \geq \epsilon/(n_{\max}(q_S + 1))$, that its running time is at most that of F plus the time of $n_{\max} q_S$ signing protocols, and that it performs at most q_S signature queries, $q_K + n_{\max} q_S$ key derivation queries, and q_H random oracle queries. □

Viewing that both security notions are essentially equivalent and that the notion of single-signer security is much easier to work with, we will stick to the latter throughout the rest of the paper. When we talk about the advantage of a forger or the security of an IBMS scheme, we implicitly mean the advantage and security under the single-signer notion.

4 Our Scheme

In this section, we present an IBMS scheme based on the GQ identity-based signature scheme [18]. To give some intuition into our scheme, we briefly recall the GQ scheme here. The key distribution center generates an RSA modulus N and exponents e, d such that $ed \equiv 1 \bmod \varphi(N)$. The master public key is the pair (N, e), while d is the master secret key. The signature on a message m by identity ID is a pair (R, s) such that $s^e \equiv R \cdot \mathrm{H}_2(ID)^c \bmod N$ where $c = \mathrm{H}_1(R \| m)$ and $\mathrm{H}_1, \mathrm{H}_2$ are random oracles. As pointed out by [17], to make an aggregate variant of a randomized signature scheme, one must find a way to "aggregate the randomness" in the different signatures. In the case of GQ signatures, this can be achieved by multiplying elements together, so that a signature by identities $ID_1, \ldots, ID_n$ is a pair (R, s) such that

$$s^e \equiv R \cdot \prod_{i=1}^{n} \mathrm{H}_2(ID_i)^c \bmod N ,$$

where R and s are the product of the R_i and s_i generated by all signers, respectively, and $c = \mathrm{H}_1(R\|m)$. Note that the combined randomness R is needed in order to compute c, which is the reason why the scheme has an interactive signing protocol.

The basic multi-signature scheme is not new: it was already present in [18] and was recently strengthened to provide robustness in [8], but was never proved secure as such. If one were to attempt such a proof, it would be complicated by the fact that, just like for other signature scheme following the Fiat-Shamir paradigm [14], simulation of signatures relies on the unpredictability of the randomness R used in the signature. In particular, for the simulator to be able to program the random oracle H_1 to simulate signature queries, the adversary's view needs to be independent of R. One way to overcome this problem [24] is by guessing, for each of the q_S signature queries, the index of a random oracle query that contains the correct R, and rewinding the adversary if the guess is incorrect. To avoid an exponential blowup of the running time, one has to restrict the multi-signature scheme to forbid concurrent signing sessions. This may be a limiting restriction viewing the inherently concurrent execution setting on the Internet. Instead, we apply a recent technique of [3] to regain provable security for concurrent protocol executions. The trick consists of letting signers commit to their randomness share R_i through a random oracle in the first round of the protocol, so that the simulator, who sees all random oracle queries, can extract these values and correctly program the random oracle before the final value of R is known to the forger.

The RSA assumption. An *RSA key generator* $\mathsf{Kg}_{\mathrm{rsa}}$ is an algorithm that generates triplets (N, e, d) such that N is the product of two large primes and $ed \equiv 1 \bmod \varphi(N)$. The advantage of A in breaking the one-wayness of RSA related to $\mathsf{Kg}_{\mathrm{rsa}}$ is defined as

$$\mathbf{Adv}^{\text{ow-rsa}}_{\mathsf{Kg}_{\mathrm{rsa}}}(\mathsf{A}) = \Pr\left[x^e \equiv y \bmod N \;;\; \begin{array}{c} (N,e,d) \xleftarrow{\$} \mathsf{Kg}_{\mathrm{rsa}} \;;\; y \xleftarrow{\$} \mathbb{Z}_N^* \;; \\ x \xleftarrow{\$} \mathsf{A}(N,e,y) \end{array} \right].$$

We say that A (t,ϵ)-breaks the one-wayness of RSA with respect to $\mathsf{Kg}_{\mathrm{rsa}}$ if it runs in time at most t and has advantage $\mathbf{Adv}^{\text{ow-rsa}}_{\mathsf{Kg}_{\mathrm{rsa}}}(\mathsf{A}) \geq \epsilon$, and we say that the RSA function associated to $\mathsf{Kg}_{\mathrm{rsa}}$ is (t,ϵ)-one-way if no algorithm A (t,ϵ)-breaks it.

The scheme. We now present our scheme in more detail. Let $l_0, l_1, l_N, \in \mathbb{N}$, and let $\mathrm{H}_0 : \{0,1\}^* \to \{0,1\}^{l_0}$, $\mathrm{H}_1 : \{0,1\}^* \to \{0,1\}^{l_1}$ and $\mathrm{H}_2 : \{0,1\}^* \to \mathbb{Z}_N^*$ be random oracles, where H_2 depends on the master public key of the scheme. Let $\mathsf{Kg}_{\mathrm{rsa}}$ be an RSA key pair generator that outputs triplets (N, e, d) such that $\varphi(N) > 2^{l_N}$ and with prime encryption exponents e of length strictly greater than $l_1 + \log_2 n_{\max}$ bits. To these, we associate the following identity-based multi-signature scheme IBMS-GQ.

Setup. The key distribution center runs $\mathsf{Kg}_{\mathrm{rsa}}$ to generate RSA parameters (N, e, d). It publishes $mpk = (N, e)$ as the master public key, and keeps the master secret key d secret.

Key derivation. On input master secret key d and signer identity ID, the key distribution center computes $x \leftarrow \mathrm{H}_2(ID)^d \bmod N$, and sends the user secret key x over a secure and authenticated channel to the signer with identity ID.

Signing. On input user secret key x_1 for identity ID_1, message m and cosigner identities $ID_2, \ldots, ID_n$, a signer proceeds as follows.

Round 1:

- Local input: x_1, $L = \{ID_1, \ldots, ID_n\}$, m
- Computation: Choose $r_1 \xleftarrow{\$} \mathbb{Z}_N^*$, compute $R_1 \leftarrow r_1^e \bmod N$ and $t_1 \leftarrow \mathrm{H}_0(R_1)$.
- Send to signer i: t_1

Round 2:

- Receive from signer i: t_i
- Send to signer i: R_1

Round 3:

- Receive from signer i: R_i
- Computation: Check that $t_i = \mathrm{H}_0(R_i)$ for all $2 \leq i \leq n$, and halt the protocol with local output $\perp$ if one of these tests fails. Otherwise, compute $R \leftarrow \prod_{i=1}^n R_i \bmod N$, $c \leftarrow \mathrm{H}_1(R \| \langle L \rangle \| m)$ and $s_1 \leftarrow r_1 x_1^c \bmod N$.
- Send to signer i: s_1

Round 4:

- Receive from signer i: s_i
- Computation: $s \leftarrow \prod_{i=1}^n s_i \bmod N$
- Local output: the signature $\sigma = (c, s)$

Verification. On input the master public key (N, e), a multiset of signer identities $L = \{ID_1, \ldots, ID_n\}$, a message m and a candidate signature (c, s), the verifier recomputes $R \leftarrow s^e \big(\prod_{i=1}^n \mathrm{H}_2(ID_i) \big)^{-c} \bmod N$. He accepts the signature as valid if $c = \mathrm{H}_1(R \| \langle L \rangle \| m)$, and rejects otherwise.

The length of a multi-signature is $l_1 + l_N$ bits, or about $160 + 1024 = 1184$ bits for typical values of the security parameter. Signing takes two exponentiations in $\mathbb{Z}_N^*$ for each signer, and verification takes a single (multi-)exponentiation, independent of the value of n. (Note that verification time is not completely independent of n due to the computation of $\prod_{i=1}^n \mathrm{H}_2(ID_i)$, but this is fast.)

While largely based on a combination of existing schemes and techniques, we believe the above scheme is important viewing the practical attractiveness of the identity-based setting in combination with compact signatures. Our scheme points out that provably secure IBMS can be achieved without the use of pairings, which is an interesting observation in its own right. Pairings have only recently been introduced to cryptography, and may therefore be at a greater risk of novel security breaches than the better-tested assumptions relating to RSA. Moreover, hardware and software implementations of RSA are ubiquitous, even in the public domain, while good pairing implementations are much harder

to find. Many companies have invested in efficient and secure implementations of RSA, and may prefer to recycle these investments in their future products.

5 Security of Our Scheme

The following theorem relates the unforgeability of our IBMS scheme to the one-wayness of the RSA problem associated to $\mathsf{Kg}_{\mathrm{rsa}}$. The proof is given below. We stress that we do not run into the key generation issues of [24] because keys are generated by the trusted center instead of by the signers themselves. Also, unlike the scheme of [24], we do allow concurrent signing sessions by reusing techniques of [3].

Theorem 2. *If the RSA function associated to* $\mathsf{Kg}_{\mathrm{rsa}}$ *is* (t', ϵ')*-one-way, then the* IBMS-GQ *scheme is* $(t, q_{\mathrm{K}}, q_{\mathrm{S}}, q_{\mathrm{H}}, n_{\max}, \epsilon)$*-secure whenever* $t' \geq 2t + (2q_{\mathrm{H}} + 2q_{\mathrm{K}} + 2q_{\mathrm{S}}(n_{\max}+1) + 2n_{\max} + 4) \cdot t_{\exp}$ *and*

$$\epsilon' \;\leq\; \frac{\epsilon^2}{16q_{\mathrm{K}}^2(q_{\mathrm{H}}+1)} \;-\; \frac{2q_{\mathrm{H}}^2 + 8n_{\max}q_{\mathrm{S}}q_{\mathrm{H}} + 8n_{\max}^2q_{\mathrm{S}}^2}{2^{l_N}} - \frac{n_{\max}q_{\mathrm{S}}}{2^{l_0}} - \frac{1}{2^{l_1}}\;, \qquad (1)$$

where $t_{\exp}$ *is the time of an exponentiation in* $\mathbb{Z}_N^*$.

To prove the above theorem, we use a variant of the Forking Lemma of Pointcheval and Stern [27] that was presented in [3]. Unlike the original Forking Lemma, this variant is easily applicable to settings other than standard signature schemes. We recall the lemma here.

Lemma 3. *Let* $q \geq 1$ *be an integer and* H *a set of size* $h \geq 2$*. Let* A *be a randomized algorithm that on input* $x, h_1, \ldots, h_q$ *returns a pair, the first element of which is an integer in the range* $0, \ldots, q$ *and the second element of which we refer to as a* side output*. Let* IG *be a randomized algorithm that we call the input generator. The accepting probability of* A*, denoted* acc*, is defined by*

$$\mathrm{acc} \;=\; \Pr\left[\, J \geq 1 \;:\; x \xleftarrow{\$} \mathsf{IG}\,;\; h_1, \ldots, h_q \xleftarrow{\$} H\,;\; (J, \sigma) \xleftarrow{\$} \mathsf{A}(x, h_1, \ldots, h_q) \,\right] .$$

The forking algorithm F_{A} *associated to* A *is the randomized algorithm that takes input* x *proceeds as follows:*

Algorithm $\mathsf{F}_{\mathsf{A}}(x)$
Pick coins ρ *for* A *at random*
$h_1, \ldots, h_q \xleftarrow{\$} H\,;\; (I, \sigma) \leftarrow \mathsf{A}(x, h_1, \ldots, h_q; \rho)$
If $I = 0$ *then return* $(0, \varepsilon, \varepsilon)$
$h'_I, \ldots, h'_q \xleftarrow{\$} H\,;\; (I', \sigma') \leftarrow \mathsf{A}(x, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q; \rho)$
If $(I = I'$ *and* $h_I \neq h'_I)$ *then return* $(1, \sigma, \sigma')$ *else return* $(0, \varepsilon, \varepsilon)$.

Let

$$\mathrm{frk} \;=\; \Pr\left[\, b = 1 \;:\; x \xleftarrow{\$} \mathsf{IG}\,;\; (b, \sigma, \sigma') \xleftarrow{\$} \mathsf{F}_{\mathsf{A}}(x) \,\right] .$$

Then

$$\mathrm{frk} \;\geq\; \mathrm{acc} \cdot \left(\frac{\mathrm{acc}}{q} - \frac{1}{h} \right) . \qquad (2)$$

We are now ready to prove the security of our IBMS scheme.

Proof (Theorem 2). Given a forger F, consider the following algorithm A. On inputs $(N, e, y), h_1, \ldots, h_{q_\mathrm{H}+1}$, algorithm A runs F on inputs $mpk = (N, e)$.

Algorithm A maintains a counter ctr_1 with initial value 0 and initially empty associative arrays $T_0[\cdot]$, $T_1[\cdot,\cdot]$, $T_2[\cdot]$. It runs F on input $mpk = (N, e)$ and answers F's oracle queries as follows.

- $\mathrm{H}_0(R)$: If $T_0[R]$ is undefined, then A chooses $T_0[R] \xleftarrow{\$} \{0,1\}^{l_0}$. It returns $T_0[R]$ to F.
- $\mathrm{H}_1(Q)$: A returns $T_1[Q]$, increasing ctr_1 and setting $T_1[Q] \leftarrow h_{ctr_1}$ if this entry is not yet defined.
- $\mathrm{H}_2(ID)$: We use Coron's technique [11] when simulating H_2 to obtain a tighter security bound. If $T_2[ID] = (b, x, X)$ then A returns X. If this entry is not yet defined, it chooses $x \xleftarrow{\$} \mathbb{Z}_N^*$ and tosses a biased coin b so that $b = 0$ with probability δ and $b = 1$ with probability $1 - \delta$. If $b = 0$, then A sets $X \leftarrow x^e \bmod N$; if $b = 1$, it sets $X \leftarrow x^e y \bmod N$. It stores $T_2[ID] \leftarrow (b, x, X)$ and returns X to F.
- Key derivation query for ID: Algorithm A looks up $T_2[ID] = (b, x, X)$, performing an additional query $\mathrm{H}_2(ID)$ if this entry is not yet defined. If $b = 0$, then A returns x; otherwise, it sets $bad_0 \leftarrow \mathtt{true}$ and aborts the execution of F returning $(0, \varepsilon)$.
- Signature query for identity ID_1, multiset of cosigners $L = \{ID_1, \ldots, ID_n\}$ and message m: Algorithm A first performs a query $\mathrm{H}_2(ID_1)$ and looks up $T_2[ID_1] = (b_1, x_1, X_1)$. If $b_1 = 0$, then A simulates signer ID_1 following the real $\mathsf{Sign}(x_1, L, m)$ algorithm using x_1 as a secret key. If $b_1 = 1$, it simulates the signing protocol as follows.
  It first chooses $t_1 \xleftarrow{\$} \{0,1\}^{l_0}$ and sends t_1 to all other cosigners. After having received $t_2, \ldots, t_n$ from all other cosigners (whose role is played by F), it chooses $c \xleftarrow{\$} \{0,1\}^{l_1}$, $s_1 \xleftarrow{\$} \mathbb{Z}_N^*$ and computes $R_1 \leftarrow s_1^e X_1^{-c} \bmod N$. If $T_0[R_1]$ has already been defined, then A sets $bad_1 \leftarrow \mathtt{true}$ and halts returning $(0, \varepsilon)$; otherwise, it sets $T_0[R_1] \leftarrow t_1$. For all $2 \le i \le n$, A looks up values R_i such that $T_0[R_i] = t_i$. If for some i multiple such values are found, A sets $bad_2 \leftarrow \mathtt{true}$ and halts returning $(0, \varepsilon)$. If for some i no such value was found then it sets $alert \leftarrow \mathtt{true}$; otherwise, it computes $R \leftarrow \prod_{i=1}^n R_i \bmod N$ and sets $T_1[R \| \langle L \rangle \| m] \leftarrow c$, or sets $bad_3 \leftarrow \mathtt{true}$ and halts with output $(0, \varepsilon)$ if this entry was already defined. It sends R_1 to all other cosigners.
  After having received $R'_2, \ldots, R'_n$ from the cosigners, A verifies that $\mathrm{H}_0(R'_i) = t_i$ for all $2 \le i \le n$. If not, it ends this signing protocol with local output $\perp$. If $R_i \neq R'_i$ for some i, then A sets $bad_2 \leftarrow \mathtt{true}$ and halts with output $(0, \varepsilon)$. If $alert = \mathtt{true}$ then it sets $bad_4 \leftarrow \mathtt{true}$ and halts with output $(0, \varepsilon)$. Otherwise, it sends s_i to all cosigners.
  After having received $s_2, \ldots, s_n$ from the cosigners, A computes $s \leftarrow \prod_{i=1}^n s_i \bmod N$ and returns the signature (c, s) to F.

Eventually, F outputs a forged signature (c, s) together with multiset of identities $L = \{ID_1, \ldots, ID_n\}$ and message m. Algorithm A computes performs additional random oracle queries $\mathrm{H}_2(ID_i)$ for $1 \le i \le n$, computes $R \leftarrow s^e \prod_{i=1}^n \mathrm{H}_2(ID_i)^{-c}$ and performs another random oracle query $\mathrm{H}_1(R \| \langle L \rangle \| m)$.

Let $U \subseteq \{ID_1, \ldots, ID_n\}$ be the uncorrupted identities in L, meaning those for which F never submitted a key derivation query. If the forgery is invalid, meaning that $\mathsf{Vf}(mpk, L, m, (R, s)) = 0$, $U = \emptyset$, or F previously made a signing query (ID, L, m), then A returns $(0, \varepsilon)$. Otherwise, algorithm A looks up $T_2[ID_i] = (b_i, x_i, X_i)$ for $1 \le i \le n$. Let $L_0 = \{ID_i : b_i = 0\}$ and $L_1 = \{ID_i : b_i = 1\}$. Since the forgery is valid, we have that

$$s^e \;\equiv\; R \cdot \prod_{i=1}^{n} X_i^c \;\equiv\; R \cdot \prod_{i=1}^{n} x_i^{ec} \cdot \prod_{i \in L_1} y^c \bmod N \;.$$

Let J be the index such that $h_J = c = T_1[R \| \langle L \rangle \| m]$. If $L_1 = \emptyset$ then A sets $bad_0 \leftarrow \mathtt{true}$ and halts with output $(0, \varepsilon)$. Otherwise, it lets $x \leftarrow \prod_{i=1}^{n} x_i$, $n_1 \leftarrow |L_1|$, and halts with output $(J, (x, c, s, n_1))$.

We want to lower-bound the probability that A produces a "useful" output, i.e. an output other than $(0, \varepsilon)$. This is exactly the accepting probability acc as defined in Lemma 3 with respect to $H = \{0,1\}^{l_1}$ and an input generator IG that returns triples (N, e, y) such that $(N, e, d) \xleftarrow{\$} \mathsf{Kg}_{\mathrm{rsa}}$ and $y \xleftarrow{\$} \mathbb{Z}_N^*$. We overload our notation to let bad_i denote the event that the flag bad_i gets set to $\mathtt{true}$ during the execution of A. We can lower-bound the accepting probability of A probability by:

$$\mathrm{acc} \;\ge\; \epsilon \cdot \Pr[\,\neg bad_0\,] - \Pr[\,bad_1\,] - \Pr[\,bad_2\,] - \Pr[\,bad_3\,] - \Pr[\,bad_4\,] \;. \quad (3)$$

First, let's take look at the factor $\Pr[\,\neg bad_0\,]$. The flag bad_0 gets raised whenever F makes a key derivation query for an identity for which $b = 1$, and if the final forgery does not contain any identities for which $b = 1$. Since the set L in the forgery must contain at least one uncorrupted identity, we have that $\Pr[\,\neg bad_0\,] \ge \delta^{q_\mathrm{K}}(1-\delta)$. This function reaches a maximum for $\delta = q_\mathrm{K}/(q_\mathrm{K}+1)$; filling in this value of δ in the above expression gives

$$\Pr[\,\neg bad_0\,] \;\ge\; \left(\frac{q_\mathrm{K}}{q_\mathrm{K}+1}\right)^{q_\mathrm{K}} \cdot \frac{1}{q_\mathrm{K}+1} \;=\; \frac{1}{q_\mathrm{K}} \cdot \left(1 - \frac{1}{q_\mathrm{K}+1}\right)^{q_\mathrm{K}+1}$$

from which we can conclude that

$$\Pr[\,\neg bad_0\,] \;\ge\; \frac{1}{4q_\mathrm{K}} \;, \quad (4)$$

because $\Pr[\,\neg bad_0\,] = 1$ if $q_\mathrm{K} = 0$, because $\Pr[\,\neg bad_0\,] \ge 1/(4q_\mathrm{K})$ for $q_\mathrm{K} = 1$, and because $(1 - 1/(q_\mathrm{K}+1))^{q_\mathrm{K}+1}$ is a monotonically increasing sequence for $q_\mathrm{K} \ge 1$.

The flag bad_1 gets raised during one of the q_S signature queries when $T_0[\cdot]$ is defined for an argument that is uniformly distributed over $\mathbb{Z}_N^*$ and that is independent from F's view. Since at any moment there are at most $q_\mathrm{H} + n_{\max} q_\mathrm{S}$ entries defined in table T_0, the probability that this happens is at most

$$\Pr[\,bad_1\,] \;\le\; \frac{q_\mathrm{S} \cdot (q_\mathrm{H} + n_{\max} q_\mathrm{S})}{2^{l_N}} \;. \quad (5)$$

The flag bad_2 only gets raised when two different entries in T_0 have the same value assigned to them. Since T_0 contains at most $q_\mathrm{H} + n_{\max} q_\mathrm{S}$ values that are all chosen uniformly at random from $\{0,1\}^{l_0}$ this happens with probability at most

$$\Pr[\, bad_2 \,] \;\leq\; \frac{(q_\mathrm{H} + n_{\max} q_\mathrm{S})^2}{2^{l_N+1}} \;. \tag{6}$$

To bound the probability that bad_3 is raised during the i-th signing query, we distinguish between the case that F "knows" R_1, meaning that it either queried $\mathrm{H}_0(R_1)$ directly, or saw R_1 as the honest signer's randomness in a previous signature query, and the case that it doesn't "know" R_1. In the latter case, F's view is independent of R, so the probability that this happens is simply given by the number of defined entries in T_1, which is at most $q_\mathrm{H} + q_\mathrm{S}$, divided by 2^{l_N}. In the former case, we cannot say that F's view is independent of R, so F may have queried $\mathrm{H}_1(R, \langle L \rangle, m)$ on purpose. Suppose F previously made a query $\mathrm{H}_0(R_1)$. Until right before this query, F's view was independent of R_1, so it had probability at most $q_\mathrm{H}/2^{l_N}$ to guess it correctly during any of its q_H queries. Likewise, the probability that A previously used the same randomness R_1 in a signature simulation is at most $q_\mathrm{S}/2^{l_N}$. In total, we have that

$$\Pr[\, bad_3 \,] \;\leq\; q_\mathrm{S} \cdot \left(\frac{q_\mathrm{H} + q_\mathrm{S}}{2^{l_N}} + \frac{q_\mathrm{H}}{2^{l_N}} + \frac{q_\mathrm{S}}{2^{l_N}} \right) \;=\; \frac{2q_\mathrm{S}(q_\mathrm{H} + q_\mathrm{S})}{2^{l_N}} \;. \tag{7}$$

Lastly, the probability that bad_4 gets set is bounded by the probability that F managed to "predict" the value of $\mathrm{H}_0(R_i)$ during one of the q_S signature protocols and for one of the at most $n_{\max}$ signers, which is

$$\Pr[\, bad_4 \,] \;\leq\; \frac{n_{\max} q_\mathrm{S}}{2^{l_0}} \;. \tag{8}$$

Combining Equations (3–8) and using $n_{\max} > 0$ gives

$$\begin{aligned}
\mathrm{acc} \;&\geq\; \frac{\epsilon}{4q_\mathrm{K}} - \frac{q_\mathrm{S}(q_\mathrm{H} + n_{\max} q_\mathrm{S})}{2^{l_N}} - \frac{(q_\mathrm{H} + n_{\max} q_\mathrm{S})^2}{2^{l_N+1}} - \frac{2q_\mathrm{S}(q_\mathrm{H} + q_\mathrm{S})}{2^{l_N}} - \frac{n_{\max} q_\mathrm{S}}{2^{l_0}} \\
&\geq\; \frac{\epsilon}{4q_\mathrm{K}} - \frac{3q_\mathrm{S}(q_\mathrm{H} + n_{\max} q_\mathrm{S})}{2^{l_N}} - \frac{q_\mathrm{H}^2 + 2n_{\max} q_\mathrm{S} q_\mathrm{H} + n_{\max}^2 q_\mathrm{S}^2}{2^{l_N+1}} - \frac{n_{\max} q_\mathrm{S}}{2^{l_0}} \\
&\geq\; \frac{\epsilon}{4q_\mathrm{K}} - \frac{q_\mathrm{H}^2 + 4n_{\max} q_\mathrm{S} q_\mathrm{H} + 4n_{\max}^2 q_\mathrm{S}^2}{2^{l_N}} - \frac{n_{\max} q_\mathrm{S}}{2^{l_0}} \;.
\end{aligned} \tag{9}$$

Now consider an algorithm B that on input (N, e, y) runs the forking algorithm $\mathsf{F}_\mathsf{A}((N, e, y))$, which with probability frk returns a tuple $(1, (x, c, s, n_1), (x', c', s', n_1'))$ with $c \neq c'$. Since these originate from valid forgeries, their values are such that

$$s^e \equiv R x^{ec} y^{cn_1} \bmod N \quad \text{and} \quad s'^e \equiv R' x'^{ec'} y^{c'n_1'} \bmod N \;.$$

The two executions of A when run by F_A are identical up to the "crucial" random oracle queries $\mathrm{H}_1(R \| \langle L \rangle \| m)$ and $\mathrm{H}_1(R' \| \langle L' \rangle \| m')$, where R, L, m and R', L', m'

are the randomness, identity sets and messages that F used in its first and second forgeries, respectively. By the construction of A, we know that the two executions of F are identical up to this query (because it was provided with the exact same input, random tape and oracle responses), so in particular we have that $R = R'$, $L = L'$ and $m = m'$. Since the entries $T_2[ID_i] = (b_i, x_i, X_i)$ for $ID_i \in L = L'$ are chosen by A at the latest at the time of the crucial hash query, we also have that $x = x'$ and $n_1 = n'_1$. Dividing and reorganizing the two equations above gives

$$(x^{c-c'} s/s')^e \equiv y^{(c-c')n_1} \bmod N .$$

Since $c \neq c' \in \{0,1\}^{l_1}$, $n_1 \leq n_{\max}$, and e is a prime of length strictly greater than $l_1 + \log_2(n_{\max})$, we have that $e > (c-c')n_1$ and therefore that $\gcd(e, (c-c')n_1) = 1$. Using the extended Euclidean algorithm, one can find $a, b \in \mathbb{Z}$ such that $ae + b(c-c')n_1 = 1$. We then have that

$$y \equiv y^{ae+b(c-c')n_1} \equiv \left(y^a \cdot (x^{c-c'} s/s')^b\right)^e \bmod N .$$

Algorithm B can therefore output $y^a \cdot (x^{c-c'} s/s')^b$ as the RSA inversion of y. The probability that algorithm B succeeds in doing so is given by

$$\begin{aligned}
\epsilon' &\geq \mathrm{frk} \\
&\geq \frac{\mathrm{acc}^2}{q_{\mathrm{H}}+1} - \frac{1}{2^{l_1}} \\
&\geq \frac{\epsilon^2}{16 q_{\mathrm{K}}^2 (q_{\mathrm{H}}+1)} - 2 \cdot \left(\frac{q_{\mathrm{H}}^2 + 4 n_{\max} q_{\mathrm{S}} q_{\mathrm{H}} + 4 n_{\max}^2 q_{\mathrm{S}}^2}{2^{l_N}} - \frac{n_{\max} q_{\mathrm{S}}}{2^{l_0}} \right) - \frac{1}{2^{l_1}}
\end{aligned}$$

where in the last step we use Equation (9) and the facts that $(a-b)^2 \geq a^2 - 2ab$ and that $0 \leq \epsilon/4 q_{\mathrm{K}} \leq 1$. The theorem follows.

We have left to show the bound for the running time t' of B. We permit ourselves to assume that (multi-)exponentiations in $\mathbb{Z}_N^*$ take time t_{exp} while all other operations take zero time. The running time of B is twice that of A, plus one multi-exponentiation mod N. The running time of A is that of the forger F plus one at most $n_{\max}+1$ multi-exponentiations plus the time needed to answer F's oracle queries. Each random oracle or key derivation query takes at most one exponentiation. A signature simulation takes at most $n_{\max}+1$ exponentiations. We therefore have that $t' = 2t + 2(n_{\max} + 2 + q_{\mathrm{H}} + q_{\mathrm{K}} + q_{\mathrm{S}}(n_{\max}+1)) \cdot t_{\mathrm{exp}}$. □

6 Alternative Implementations

Our scheme is based on the GQ signature scheme [18], but our techniques can be applied to other identity-based signature schemes following the Fiat-Shamir paradigm as well. In particular, one can obtain efficient IBMS schemes based on RSA from [28], based on factoring from [14,13,25,26], and based on pairings from [19,9,29]. An extensive overview of the security properties of these schemes as identity-based signature schemes can be found in [2].

Acknowledgments

Mihir Bellare was supported by NSF grant CNS-0524765, a gift from Intel Corporation, and NSF CyberTrust project "CT-ISG: Cryptography for Computational Grids". Gregory Neven is a Postdoctoral Fellow of the Flemish Research Foundation (FWO – Vlaanderen), and was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

References

1. K. Barr and K. Asanovic. Energy aware lossless data compression. In *MobiSys 2003*, pages 231–244. ACM Press, 2003.
2. M. Bellare, C. Namprempre, and G. Neven. Security proofs for identity-based identification and signature schemes. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 268–286. Springer-Verlag, 2004.
3. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM CCS 06*. ACM Press, 2006.
4. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM Press, 1993.
5. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer-Verlag, 2003.
6. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer-Verlag, 2003.
7. C. Castelluccia, S. Jarecki, J. Kim, and G. Tsudik. A robust multisignatures scheme with applications to acknowledgment aggregation. In C. Blundo and S. Cimato, editors, *SCN 2004*, volume 3352 of *LNCS*, pages 193–207. Springer-Verlag, 2005.
8. C. Castelluccia, S. Jarecki, J. Kim, and G. Tsudik. Secure acknowledgment aggregation and multisignatures with limited robustness. *Computer Networks*, 50(10):1639–1652, 2006.
9. J. C. Cha and J. H. Cheon. An identity-based signature from gap Diffie-Hellman groups. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 18–30. Springer-Verlag, 2003.
10. X. Cheng, J. Liu, and X. Wang. Identity-based aggregate and verifiably encrypted signatures from bilinear pairing. In O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar, and C. J. K. Tan, editors, *Computational Science and Its Applications ICCSA 2005*, volume 3483 of *LNCS*, pages 1046–1054. Springer-Verlag, 2005.
11. J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer-Verlag, 2000.
12. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 130–144. Springer-Verlag, 2003.
13. U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

14. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, 1987.
15. S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. `http://eprint.iacr.org/`, 2006.
16. D. Galindo, J. Herranz, and E. Kiltz. On the generic construction of identity-based signatures with additional properties. To appear in *ASIACRYPT 2006*, *LNCS*. Springer-Verlag, 2006.
17. C. Gentry and Z. Ramzan. Identity-based aggregate signatures. In M. Yung, editor, *PKC 2006*, volume 3958 of *LNCS*, pages 257–273. Springer-Verlag, 2006.
18. L. C. Guillou and J.-J. Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In S. Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 216–231. Springer-Verlag, 1990.
19. F. Hess. Efficient identity based signature schemes based on pairings. In K. Nyberg and H. M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 310–324. Springer-Verlag, 2003.
20. K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, 71:1–8, 1983.
21. A. Joux. A one round protocol for tripartite Diffie-Hellman. In *Algorithmic Number Theory Symposium – ANTS IV*, volume 1838 of *LNCS*, pages 385–394. Springer-Verlag, 2000.
22. S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*. Springer-Verlag, 2006.
23. A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 74–90. Springer-Verlag, 2004.
24. S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures. In *ACM CCS 01*, pages 245–254. ACM Press, 2001.
25. K. Ohta and T. Okamoto. A modification of the Fiat-Shamir scheme. In S. Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 232–243. Springer-Verlag, 1990.
26. H. Ong and C.-P. Schnorr. Fast signature generation with a Fiat Shamir–like scheme. In I. Damgård, editor, *EUROCRYPT'90*, volume 473 of *LNCS*, pages 432–440. Springer-Verlag, 1990.
27. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.
28. A. Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and D. Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer-Verlag, 1985.
29. X. Yi. An identity-based signature scheme from the Weil pairing. *IEEE Communications Letters*, 7(2):76–78, Feb. 2003.

Improved Efficiency for Private Stable Matching

Matthew Franklin, Mark Gondree, and Payman Mohassel

Department of Computer Science
University of California, Davis
{franklin, gondree, mohassel}@cs.ucdavis.edu

Abstract. At Financial Crypto 2006, Golle presented a novel framework for the privacy preserving computation of a stable matching (stable marriage). We show that the communication complexity of Golle's main protocol is substantially greater than what was claimed in that paper, in part due to surprising pathological behavior of Golle's variant of the Gale-Shapley stable matching algorithm. We also develop new protocols in Golle's basic framework with greatly reduced communication complexity.

Keywords: stable matching, stable marriage, Gale-Shapley, privacy-preserving protocols, secure multiparty computation, passive adversaries.

1 Introduction

Efficient stable matching (stable marriage) algorithms are used in a wide variety of practical settings, including the well-known example of matching U. S. medical school graduates to hospitals, for their residencies. Gusfield and Irving [14] have written a good overview of stable matching algorithms.

Golle [12] argues persuasively that efficient privacy-preserving protocols for stable matching could have great practical benefit. In fact, Golle goes on to develop just such a protocol. We find the basic framework of his approach to be quite appealing, and worthy of further examination. In his framework, some number of honest-but-curious "matching authorities" (MAs) receive encrypted preference lists from the participants, and then execute a variant of the classic Gale-Shapley algorithm that has been specially tuned for concealment. The main cryptographic tools in Golle's protocol are threshold homomorphic encryption and re-encryption mixnets.

Our first contribution is to show that Golle's protocol has substantially greater communication complexity than what was reported in the original paper. For example, the total communication is $O(tn^5)$ ciphertexts instead of $O(n^3)$ ciphertexts as reported (where the number of matching authorities is t, and the number of participants is $O(n)$). This is due in part to a surprising anomaly in Golle's variant of the Gale-Shapley algorithm that requires it to run for more rounds than Golle's analysis suggests (quadratic rather than linear in the number of participants).

Our second contribution is to design new privacy preserving protocols in Golle's basic framework with reduced communication complexity (under similar cryptographic assumptions). Our protocol in Section 4.2 has improved communication complexity when there are an arbitrary number t of matching authorities. Our protocol in Section 5

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 163–177, 2007.

reduces the communication complexity even further when there are exactly two matching authorities. One way we achieve our improved efficiency is by designing our own variant of Gale-Shapley that is "tuned for concealment" but with better convergence properties than Golle's variant. The following table summarizes the efficiency of these new protocols and our new analysis of Golle's protocol.

Section	Protocol	MAs	Total Work	Total Communication	Round Complexity
4.1	Golle's	t	$O(n^5)$	$O(tn^5)$	$\widetilde{O}(n^3)$
4.2	Ours	t	$O(n^4\sqrt{\log n})$	$O(tn^3)$	$\widetilde{O}(n^2)$
5	Ours	2	$O(n^4)$	$O(n^2)$	$\widetilde{O}(n^2)$

The organization of the rest of the paper is as follows. Preliminary notions (models, definitions, primitives) are given in Section 2. In Section 3, we discuss the Gale-Shapley stable matching algorithm, with a careful analysis of Golle's variant and our new variant. In Section 4, we present and analyze Golle's protocol and our protocol for the case with multiple matching authorities. In Section 5, we describe a protocol for the case of just two matching authorities.

2 Preliminaries

2.1 The Stable Marriage Problem

We consider the formulation of the stable matching problem with complete preference lists (every man ranks every woman, and vice-versa) and one-to-one matchings. Due to its simplicity, and the fact that other variants of the stable matching problem can be reduced to this formulation, it is a particularly attractive version with which to work. The problem is as follows. Consider two sets, one of n men and one of n women. Every man ranks the n women, and every woman ranks the n men. A matching, or *marriage*, is a bijection between the sets. A matching is *stable* if there is no unmatched man and woman who rank each other higher than their own spouses. The stable marriage problem is, given the preference lists of n men and n women, to find a stable marriage (there is always one, and there may be several). In Section 3, we discuss algorithms for the stable marriage problem.

2.2 Models and Definitions

We adopt the same network model as Golle [12]. At the start of the protocol, each player sends a single encrypted message (derived from his or her preferences) to two or more *matching authorities* (MA's). These matching authorities execute a synchronous protocol among themselves to compute the stable matching.

For simplicity (and fairness of comparison), our security model is the same as that considered by Golle [12]. Specifically, we consider a *passive* adversary, meaning an adversary with passive control over any of the players (men and women) and passive control over all but one of the matching authorities. More precisely, our security guarantees

hold for any adversary in the intersection of the adversarial models of the primitives we use. Our guarantees are in relation to the following security notion, due to Golle [12]: a protocol is a *private stable matching protocol* if it outputs a stable match and reveals no other information to a passive adversary than what she can learn from the matching and from the preferences of the participants she controls.

In all our constructions, we compose protocols that are private with respect to passive adversaries, and make use of composition theorems that guarantee security under composition. These theorems enable us to claim our protocols private against a passive adversary as long as our subprotocols are private. Please see [3,10] for more details.

Encryption. Unless otherwise stated, we let E denote the encryption function for a threshold public-key encryption scheme that is additively homomorphic, such as a threshold version [7,6] of the Paillier encryption scheme [23]. The matching authorities are the joint holders of the decryption key, such that only a quorum of all parties can decrypt.

Notation. We use the following asymptotic notation: $o(f)$ denotes that the asymptotic upper bound f is not tight; $\Omega(f)$ denotes that the asymptotic lower bound f is tight; and $\widetilde{O}(f)$ denotes the asymptotic upper bound $O(f)$, ignoring $\mathrm{polylog}(f)$ factors. We denote the XOR operation between two equal-length bit strings a, b by $a \oplus b$. In Section 2.3 below, unless otherwise noted, "poly-log complexity" is in reference to the security parameter for each primitive.

2.3 Primitives

Re-encryption Mix Network. In our application, when we say the authorities *mix* some (Paillier) ciphertexts, we mean the authorities run a re-encryption mix network [22,15], permuting the ciphertexts according to a secret permutation known to none of the individual authorities. Since we consider a passive adversary, n ciphertexts can be mixed by t mixing authorities in constant round and $O(n)$ time, taking advantage of parallel mixing techniques [13]. The total communication complexity of the parallel mixnet is, like a serial mixnet, $O(tn)$ ciphertexts.

Private Oblivious Equality Test. Let $E(m_1), E(m_2)$ be two Paillier ciphertexts. Define $\mathsf{EQTEST}(E(m_1), E(m_2)) = b$ where $b = 1$ if $m_1 = m_2$ and $b = 0$ otherwise. EQTEST is a (chooser private) oblivious test of plaintext equality [16,20] if it reveals the output to the joint holders of the decryption keys, without revealing any other information to any other parties.

MPC Private Equality Test. Let $E(m_1), E(m_2)$ be two Paillier ciphertexts. Define $\mathsf{EEQTEST}(E(m_1), E(m_2)) = E(b)$ where $b = 1$ if $m_1 = m_2$ and $b = 0$ otherwise. $\mathsf{EEQTEST}$ is the secure multiparty computation of the equality test if our parties learn the output $E(b)$, but no additional information about the plaintexts m_1, m_2. [4,17] both give constant-round protocols with poly-log communication complexity for computing this function, either of which are adaptable to our setting (*i.e.*, threshold, additively homomorphic ciphertexts).

Private Oblivious Value Comparison. Let $E(m_1), E(m_2)$ be two Paillier ciphertexts. Define $\mathsf{COMPARE}(E(m_1), E(m_2)) = b$ where $b = 1$ if $m_1 < m_2$ and $b = 0$ otherwise. For our purposes, we have $0 \leq m_1, m_2 \leq n$. Golle instantiates this primitive by preparing $n-1$ ciphertexts, $D_1, \ldots, D_{n-1}$ where $D_i = E(m_1 - m_2 - i)$; mixing these n ciphertexts; and finally running n parallel instances of $\mathsf{EQTEST}(E_i, E(0))$. If $m_1 < m_2$ then, for some $0 < i \leq n$, one of these instances returns 1. Otherwise, all instances return 0.

MPC Private Value Comparison. Let $E(m_1), E(m_2)$ be two Paillier ciphertexts. Define $\mathsf{ECOMPARE}(E(m_1), E(m_2)) = E(b)$ where $b = 1$ if $m_1 < m_2$ and $b = 0$ otherwise. $\mathsf{ECOMPARE}$ is the secure multiparty computation of the less-than function if our parties learn the output $E(b)$, but no additional information about the plaintexts m_1, m_2. [17,5] both give constant-round protocols with poly-log communication complexity for computing this function, either of which are adaptable to our setting.

Private Reduction of a Secret Modulo a Public Integer. Let $E(a)$ be a Paillier ciphertext, and q be an integer. Define $\mathsf{MOD}(E(a), q) = E(a \bmod q)$. MOD is the secure multiparty computation of the modular function if our parties learn the output, but no additional information about the plaintext integer a. [1,17] both give protocols with poly-log communication complexity for computing this function, either of which are adaptable to our setting. The former has a poly-log round complexity, the latter is constant-round.

SPIR with Sublinear Communication Complexity. Let δ be a database with N elements, indexed $\{0, \ldots, N-1\}$. Let $\mathsf{SPIR}^{\delta}_{m}(\mathbf{b}_1, \ldots, \mathbf{b}_\ell)$ represent Stern's symmetric private information retrieval protocol [24]. As in any PIR protocol, a chooser holds a secret index i and, at the end of the protocol, the chooser learns the element of the database at index i, while the database learns nothing about which index was accessed. Additionally, the chooser learns nothing about any of the other database elements (thus, symmetry). In $\mathsf{SPIR}^{\delta}_{m}$, the index i is encoded following a standard trick, due to Kushilevitz and Ostrovsky [18]. The database is imagined as a series of m sized buckets (the first m entries in the first bucket, and so on). If element i is the jth element in one of these buckets, then $b_{1,j} = E(1)$ and $b_{i,k} = E(0)$ for all $k \neq j$. Define $\mathbf{b}_1 = (b_{1,1}, \ldots, b_{1,m})$. We recurse, imagining the collection of former buckets as, themselves, a series of m sized buckets. The output of the protocol must be decrypted by the chooser ℓ times, to recover the element at index i. With $m = N^{1/\ell}$ and $\ell = O(\sqrt{\log N})$, the protocol has total computational complexity $O(N\sqrt{\log N})$ and total communication complexity $2^{O(\sqrt{\log N})}$. Since we consider passive adversaries, we do not include Stern's interactive zero-knowledge proofs showing the indices are well-formed as a part of $\mathsf{SPIR}^{\delta}_{m}$.

Private Table Read/Write Protocols. Initially, party A holds i_A and $L_A[1...n]$, and party B holds i_B and $L_B[1...n]$. In other words, parties share the index $i = i_A \oplus i_B$, and the array L such that $L[j] = L_A[j] \oplus L_B[j]$ for $1 \leq j \leq n$. For our applications, we assume that the indices and elements of the array are k-bit integers, for some k. Naor and Nissim [21] design protocols for both reading and writing to the table in this setting, requiring $O(\text{polylog}(n))$ communication. Their protocols use an *oblivious transfer* protocol as their main building block. The read protocol returns $R \oplus L[i]$ to A

and R to B, where R is a random k-bit integer. In the oblivious write protocol (writing a shared value v), each party will obtain new shares of L such that $L[i] = v$.

Yao's Garbled Circuit Protocol. *Yao's garbled circuit* protocol [25] is the first general purpose secure two-party protocol. In this protocol, parties compute a functionality using the circuit for that functionality. Please see [19] for a detailed description of Yao's protocol. The protocol runs in a constant number of rounds, and has a communication and computation complexity that is linear in the size of the circuit. We use Yao's garbled circuit to design portions of our protocols. Therefore, we sometimes need to switch from a different setting to Yao's garbled circuit setting, and back to the original one. Specifically, in Section 5, we need to switch to Yao's garbled setting from a setting where inputs are shared using XOR sharing, and have the parties share the final output of the circuit using an XOR sharing. This can be done by adding small additional circuitry to the original circuit. This additional circuitry will not affect the complexity of the circuit size or the protocol.

3 Stable Marriage Algorithms

In this section, we will first briefly describe the Gale-Shapley algorithm, and then take a closer look at Golle's variant of Gale-Shapley. We explain why the complexity of this variant is a factor of n more than what was claimed in [12]. Finally, we design our own variant of the Gale-Shapley algorithm, in which we avoid the factor of n increase in the complexity while preserving the useful properties we need for a secure implementation. This new variant is what we use in Sections 4 and 5 to design more efficient *private stable matching protocols*.

3.1 The Gale-Shapley Algorithm

We review the well-known algorithm of Gale and Shapley [9], not only because of its general importance but because the private stable matching protocols presented later are, in fact, simulations of variants of the Gale-Shapley algorithm.

The Gale-Shapley algorithm considers a series of proposals made by men, round-by-round. Whenever a proposal is accepted, the couple is considered engaged. If a man is not engaged, he is considered free. The algorithm proceeds as follows. If there are any free men, select one at random (call him A). A proposes to the woman he ranks highest among those to whom he has not yet proposed (call her B). If B is free, she accepts and the pair are considered engaged. If B is engaged to some A' and she ranks A' ahead of A, then B and A' remain engaged and A remains free. If B is engaged to A' and she ranks A' below A, then B and A become engaged and A' becomes free.

After $O(n^2)$ proposals, all participants will be engaged and we will have found a stable marriage. In fact, the marriage we find is men-optimal. Due to symmetry, it is clear we could run the algorithm to find a marriage that is women-optimal. For more on the Gale-Shapley algorithm, the interested reader is referred to the treatment of the subject by Gusfield and Irving [14]. One important note, however, is that by observing the proposals, acceptances, and rejections round-by-round, one can (for some problem instances) reconstruct the entire preference lists of all participants.

3.2 Golle's Variant of Gale-Shapley

In this section we describe Golle's variant of the classic Gale-Shapley algorithm, explain its suitability for implementation as a private stable matching protocol, and present our new complexity analysis. Consider Gale-Shapley's algorithm where there are n real women $B_1, \ldots, B_n$, and n real men $A_1, \ldots, A_n$. In Golle's variant, n "fake" men, $A_{n+1}, \ldots, A_{2n}$ are introduced (no fake women are defined). The preferences of fake men are not important, and can be chosen arbitrarily. The preferences of women need to be augmented to account for the fake men introduced. It is only important that each woman ranks the fake men lower than any real ones.

In what follows, let $\mathcal{F}_k$ and $\mathcal{E}_k$ denote the sets of free and engaged men in round k of the algorithm, respectively. Initially, all the real men are free and all the fake men are engaged (in an arbitrary way). The algorithm proceeds as follows:

For $k = 1$ to R:

While $\mathcal{F}_k$ is non-empty:
- Randomly select a man A from $\mathcal{F}_k$.
- A proposes to woman B, the woman he ranks highest among the women to whom he has never proposed before.
- Note that woman B is always already engaged to some man, A'. This is resolved in the following manner.
  * If B ranks A higher than A', she breaks her engagement to A' and becomes engaged to A. Man A is removed from set $\mathcal{F}_k$ and added to $\mathcal{E}_k$, whereas man A' is removed from $\mathcal{E}_k$ and added to $\mathcal{F}_{k+1}$.
  * If B ranks A behind A', she stays engaged to A'. Man A is removed from set $\mathcal{F}_k$ and added to set $\mathcal{F}_{k+1}$.
- When $\mathcal{F}_k$ is empty, let $\mathcal{E}_{k+1} = \mathcal{E}_k$.

Note that the above algorithm has some nice properties for designing a secure stable matching protocol. For instance, all n women are always engaged to some man. During round k, the number of engaged men is always $|\mathcal{E}_k| = n$. Every time a new proposal is made, the cardinality of $\mathcal{F}_k$ decreases by one, the cardinality of $\mathcal{F}_{k+1}$ increases by one and the cardinality of $\mathcal{E}_k$ is unchanged.

The algorithm, ends after R iterations. In [12], it is claimed that $R = n$ is enough to reach a stable matching. We observe that this is not the case, and for some inputs, $\Omega(n^2)$ iterations are necessary to achieve a stable matching. This implies that proposition 1, as stated in [12], is incorrect. A problem instance explored in Gusfield and Irving [14, pg15] is one such counterexample, and is presented in Appendix A.

The intuition behind this inefficiency is that, for some inputs, there may be many rounds where most of the proposals are made by fake men. Such proposals do not help the real men move forward in their preference lists, and hence do not help them reach a stable matching. This observation implies a factor of n increase in Golle's algorithm, and the same increase in the communication complexity of his privacy preserving stable matching protocol.

Claim. The algorithm of Section 3.2 (with $R = n^2$) produces a stable matching among the n men and n women. That is, the algorithm is correct. This claim replaces Proposition 1 of [12].

Proof. After $n^2 - n + 1$ proposals from real men, we will have a stable marriage (from the same counting argument used to show the correctness of the Gale-Shapley algorithm). Until a stable marriage is reached, some real man will be free. So, after n^2 rounds, real men will have made at least n^2 proposals and, thus, the algorithm outputs a stable marriage. The minimal number of rounds required for correctness is less, but is $\Omega(n^2)$ (see Appendix A).

3.3 Our New Variant of Gale-Shapley

Here, we describe our variant of Gale-Shapley which improves on the complexity of Golle's variant by a factor of n and which also has nice properties for designing a private matching protocol.

Once again, as with Gale-Shapley's algorithm, there are n real men and n real women with their corresponding preference lists. We add n fake men and n fake women to this setting (note that Golle's variant did not include fake women). Thus, we have: real men $\{A_1, \ldots, A_n\}$, fake men $\{A_{n+1}, \ldots, A_{2n}\}$, real women $\{B_1, \ldots, B_n\}$, and fake women $\{B_{n+1}, \ldots, B_{2n}\}$. Preference lists are adjusted in the following way.

	Preference lists
Real men	([actual preference list], [$B_{n+2}, \ldots, B_{2n}$, in any order])
Fake men	([$B_{n+2}, \ldots, B_{2n}$, in any order], B_{n+1}, [$B_1, \ldots, B_n$, in any order])
Real women	([actual preference list], [$A_{n+1}, \ldots, A_{2n}$, in any order])
Fake women	([$A_{n+1}, \ldots, A_{2n}$, in any order], [$A_1, \ldots, A_n$, in any order])

As before, set $\mathcal{F}_k$ contains the free men in round k. The algorithm works as follows.

Initialization:

- $\mathcal{F}_1 = \{A_1\}$ (man A_1 is free).
- $\{A_2, \ldots, A_n\}$ are engaged to $\{B_{n+2}, \ldots, B_{2n}\}$, respectively.
- $\{A_{n+1}, \ldots, A_{2n}\}$ are engaged to $\{B_1, \ldots, B_n\}$, respectively.

For $k = 1$ to R:

- The free man A in $\mathcal{F}_k$ proposes to B, the next woman in his preference list to whom he has not yet proposed.
- Let A' denote the man to whom B is already engaged.
  * If B ranks A higher than A', she breaks her engagement to A' and becomes engaged to A. Let $\mathcal{F}_{k+1} = \{A'\}$.
  * If B ranks A lower than A', she stays engaged to A'. Let $\mathcal{F}_{k+1} = \{A\}$.

Note that in each iteration, exactly one proposal is made. The number of free men in each round is $|\mathcal{F}_k| = 1$. As we will show next, the above algorithm reaches a stable

matching in at most $2n^2$ iterations. In the matching reached, all the real men are engaged to real women and all the fake men to fake women.

Claim. Once a fake man proposes to fake woman B_{n+1}, we have reached a stable matching. In this stable matching, real men are engaged to real women and fake men are engaged to fake women.

Proof. Note that woman B_{n+1} is the n^{th} preference of all the fake men. Therefore, when a fake man proposes to B_{n+1}, he has already proposed to the other $n-1$ fake women in his list and has been rejected by them at some point during the execution of the protocol. This implies that all the other $n-1$ fake women were or became engaged to other fake men (fake women rank fake men higher than real men). This, in turn, implies that all the real women are engaged to real men.

The argument for having reached a stable matching is similar to the one for the original Gale-Shapley algorithm. Particularly, lets assume that real man A prefers woman B to woman B', to whom he is engaged. Then, B must have rejected A at some point during the execution. But, this implies that B was or became engaged to a man she prefers to A. So B cannot prefer A to her current match. This further implies that there are no unstable matches.

It is easy to verify that before $2n^2$ proposals, at least one fake man will have proposed to B_{n+1}. Therefore, based on the above claim, we reach a stable matching in at most $R = 2n^2$ steps.

For the secure implementation of the above algorithm, it is useful to run the algorithm for the same number of rounds for all inputs (e.g. $R = 2n^2$). This will help us avoid leaking the number of proposals necessary to reach a stable matching for a specific input. But, note that in the above algorithm, once a fake man proposes to woman B_{n+1}, no free man will remain and the algorithm has to end. A simple fix is to add an extra fake man A_{2n+1}, and initially let him be engaged to woman B_{n+1}. The algorithm runs exactly as before and once a fake man proposes to woman B_{n+1}, the same claims as above are true. The only advantage is that we will always have a free man who will propose next. This is a useful invariant for the secure implementation we give in Section 4.2.

4 Privacy Preserving Stable Marriage Protocols

In this section, we present the privacy preserving implementation of the two Gale-Shapley variants presented earlier: one for Golle's (modified) variant in Section 4.1, and one for our new variant in Section 4.2.

4.1 Privacy Preserving Protocol for Golle's Variant of Gale-Shapley

In this section, we briefly present the implementation of Golle's (modified) variant of Gale-Shapley as a private stable matching protocol. This section will also provide a basis for comparison with the secure variant in Section 4.2.

Protocols for the Implementation. The following are used in the secure implementation of Golle's variant given at the end of this section. Many will be of use later, in Section 4.2. Slight modifications to some protocols were necessary due to the observations from Section 3.2.

Notation. Let $r_{i,j} \in \{0, \ldots, n-1\}$ be the rank given to real woman B_j by man A_i. Let $s_{j,i} \in \{0, \ldots, n-1\}$ be the rank given to real man A_i by woman B_j. Our convention is that the highest possible rank is 0, and the lowest is $n-1$.

Bids. Define the (free) *bid* for man A_i as $W_i = \langle E(i), \mathbf{a}_i, \mathbf{v}_i, \mathbf{q}_i, E(\rho)\rangle$, where $\mathbf{a}_i = (E(r_{i,1}), \ldots, E(r_{i,n}))$, $\mathbf{q}_i = (E(s_{1,i}), \ldots, E(s_{n,i}))$, and $\mathbf{v}_i = (E(1), \ldots, E(n))$. Initially, $\rho = 0$.

Engaged Bids. The *engaged bid* $\langle W_i, E(j), E(s_{j,i})\rangle$ denotes that man A_i is engaged to woman B_j. Let $\mathcal{F}_k$ and $\mathcal{E}_k$ denote the sets of free and engaged bids in round k of the algorithm, respectively. When we *mix the bids*, the t matching authorities mix each of these sets separately.

Input Submission and Initialization. Each man A_i initially sends his preference list $\langle E(r_{1,i}), \ldots, E(r_{n,i})\rangle$ and each woman B_j sends her list $\langle E(s_{j,1}), \ldots, E(s_{j,n})\rangle$ to the matching authorities. The matching authorities jointly create the preferences for the fake men $A_{n+1}, \ldots, A_{2n}$ and augment the women's preference lists with the fake men. The matching authorities generate the $2n$ bids for $A_1, \ldots, A_{2n}$ and the n engaged bids to denote the engagement of A_{n+j} to woman B_j. We add the n engaged bids to $\mathcal{E}_1$, and the n free bids to $\mathcal{F}_1$.

Breaking an Engagement. Let $\langle W_i, E(j), E(s_{j,i})\rangle$ be an engaged bid. We break this engagement by discarding $E(j), E(s_{j,i})$ and keeping W_i. We also "safely" update $E(\rho)$ by incrementing it by the value $1-b$, where $E(b) = \mathsf{EEQTEST}(E(\rho), E(n-1))$, using Paillier's additive homomorphism (ie, we multiply $E(\rho)$ by $E(1)/E(b)$). That is, we obliviously increment the next desired rank ρ when it is less than $n-1$ and, otherwise, we do not. This is a modification from the presentation in [12]. If we did not increment safely, the new n^2 loop bound generates the possibility that we may increment some man's ρ more than n times which would lead, in a sense, to a pointer error.

Find a Conflicting Bid. Given a newly created engaged bid $\langle W_i, E(j), E(s_{j,i})\rangle$ there will be exactly one existing engaged bid that conflicts. That is, there is some engaged bid $\langle W_{i'}, E(j'), E(s_{j',i'})\rangle \in \mathcal{E}_k$ where $j = j'$. We can find this by preparing the set $\{E(j') \mid \langle W_{i'}, E(j'), E(s_{j',i'})\rangle \in \mathcal{E}_k\}$, mixing the n ciphertexts in this set, and then performing n parallel instances of $\mathsf{EQTEST}(E(j), E(j'))$ for each $E(j')$ in the mixed set.

Resolve a Conflict. Given two random conflicting engaged bids, $\langle W_i, E(j), E(s_{j,i})\rangle$ and $\langle W_{i'}, E(j), E(s_{j,i'})\rangle$, we determine the "winner" and "loser" of the conflict by doing the following. Jointly compute $b = \mathsf{COMPARE}(E(s_{j,i}), E(s_{j,i'}))$. If $b = 1$ then woman j prefers man i' over man i and, thus, we call the first engaged bid the "loser." Otherwise, we call the second engaged bid the "loser." We call the remaining bid the "winner."

Summary of the Privacy Preserving Implementation. The following algorithm, with $R = n^2$, summarizes the secure implementation of Golle's variant of Gale-Shapley. How to process the submitted inputs, initialize the data structures, find a conflicting

bid, resolve the conflict, and break an engagement are explained in Section 4.1. We have, however, omitted the details of some steps, such as internal bid mixing, opening a bid , and some others. We refer the reader to Golle's paper [12] for those details we have omitted.

Briefly, when a bid W_i is "opened," the matching authorities jointly determine $E(j)$ (the woman at rank ρ on man A_i's preference list) and her preference $E(s_{j,i})$ for A_i, without learning anything about ρ, j or i.

Input submission and Initialization
For $k = 1$ to R:

While F_k is non-empty:
1. Randomly select a bid W_i from $\mathcal{F}_k$.
2. Open W_i to recover $E(j), E(s_{j,i})$
3. Form the engaged bid $\langle W_i, E(j), E(s_{j,i})\rangle$
4. Find the conflicting engaged bid, $\langle W_{i'}, Ej, E(s_{j,i'})\rangle$
5. Mix these two engaged bids
6. Resolve the conflict to find the "winner" and "loser"
7. Break the engagement for the loser and add this free bid to $\mathcal{F}_{k+1}$
8. Add the winner to $\mathcal{E}_k$
9. Mix all the bids, and perform internal bid mixing
10. If $\mathcal{F}_k$ is empty, let $\mathcal{E}_{k+1} = \mathcal{E}_k$

Announce stable marriage

At step $k = n^2$, all data is discarded, save the ciphertext pairs $(E(i), E(j))$ from each engaged bid in $\mathcal{E}_{n^2}$. These are (publicly or privately) announced to each participant.

Complexity Analysis. The work and communication complexity is dominated by running the 3 re-encryption mix networks in steps 4, 5, and 9 — specifically, the mixnet in step 9. This re-encryption mix network is run on $2n$ bids, n times each round (since $|\mathcal{F}_k| = n$ at the start of each round). Furthermore, each bid contains $O(n)$ ciphertexts. Thus, each of the t authorities does $O(n^5)$ total work. The total communication complexity is $O(tn^5)$ ciphertexts. This differs from Golle's $O(n^3)$ analysis in [12], which claims the bid size to be constant, claims $R = n$ instead of $R = n^2$, and omits t as a factor impacting the number of messages passed. For similar reasons, the round complexity is now $O(n^3\text{polylog}(n))$, instead of $O(n^2\text{polylog}(n))$ as claimed in [12].

Claim. The protocol of Section 4.1 is a private stable matching protocol, assuming Paillier encryption is semantically secure and the underlying re-encryption mix network is private. When t matching authorities participate, the protocol's total communication complexity is $O(tn^5)$ ciphertexts. This claim replaces Propositions 2 and 3 of [12].

Proof (sketch). The correctness of the algorithm from Claim 3.2 shows the protocol outputs a stable matching. To show the protocol is private, we direct the reader to the proof sketch of Proposition 3 in [12]; our modifications to the protocol do not impact the proof that a passive adversary learns no additional information during the protocol's execution. The complexity analysis is shown above.

4.2 Privacy Preserving Protocol for Our New Variant of Gale-Shapley

To implement our new variant of Gale-Shapley securely, we must modify the initialization procedure of Golle's secure protocol to accommodate the addition of fake women. Furthermore, we present a new procedure to open a bid with the aid of a database that holds the participants' encrypted preference lists. By removing the preference lists from the bids themselves, we make our bids constant-sized. Now, we define a bid W_i for man A_i as $\langle E(i), E(\rho)\rangle$. We assume one of the t matching authorities plays the role of the database.

Protocols for the Implementation. From Section 4.1, the definition for an engaged bid and the procedures for finding a conflict, breaking an engagement, and resolving a conflict remain the same for the secure implementation of the new variant of Gale-Shapley given at the end of this section. The following procedures are also used.

Input Submission. As before, each woman sends her preference list $\mathbf{q}_i$. Similarly, each man submits a vector $\mathbf{a}_i$, but the vector encodes his preference list in a new way. Now, man A_i defines $\mathbf{a}_i = (E(a_{i,1}), \ldots, E(a_{i,n}))$, where $a_{i,j} \in \{1, \ldots, n\}$ is the index of the woman to whom he has given rank $j-1$.

Initialization. The matching authorities generate the free bid for man A_1 and the engaged bids for man A_i, for $i \neq 1$. The preference lists for the $n+1$ fake men and n fake women are generated, and the preference lists for the real men and women are augmented, according to the rules above. Let one matching authority collect and organize these lists, and call this authority the database δ. Let $\delta = [(\mathbf{a}_1, \mathbf{q}_1), \ldots, (\mathbf{a}_{2n}, \mathbf{q}_{2n})]$. Thus $\delta[4n(i-1)+(j-1)] = E(a_{i,j})$ and $\delta[4n(i-1)+(j-1)+2n] = E(s_{j,i})$, for $i, j \leq 2n$.

Open a Bid. Given a free bid $\langle E(i), E(\rho)\rangle$, we must recover $E(j)$ (the encrypted index of the woman at rank ρ on man A_i's preference list) and $E(s_{j,i})$. It happens that $E(j)$ is located at $\delta[4n(i-1)+(\rho-1)]$ and $E(s_{j,i})$ is located at $\delta[4n(i-1)+(j-1)+2n]$. We can calculate $E(4n(i-1)+\rho-1)$ using the Paillier additive homomorphism, given $E(i)$ and $E(\rho)$. We can recover $E(j)$ by accessing the database at this secret index, using the protocols below. Similarly, given $E(j)$ we can calculate $E(4n(i-1)+(j-1)+2n)$ and, again, recover $E(s_{j,i})$ by accessing the database at this secret index.

Access the Database at a Secret Index. Given $E(x)$, we can generate a series of indices $\mathbf{b}_1, \ldots, \mathbf{b}_\ell$ which singulate the element at index x using the index conversion procedure below, without learning anything about index x. Then let $y = \mathsf{SPIR}^{\delta}_{m}(\mathbf{b}_1, \ldots, \mathbf{b}_\ell)$. We jointly decrypt y, ℓ times, to recover $\delta[x]$. For the values of m and ℓ indicated in Section 2.3, this joint decryption takes $O(\sqrt{\log n})$ rounds, passing a $2^{O(\sqrt{\log n})}/2^i$ size message between t authorities on round i, yielding a total communication complexity of $o(tn)$. After this procedure, the entire database should re-encrypt all of its entries. The total computational complexity of this database access is $O(n^2\sqrt{\log n})$.

Secure Index Conversion. Given $E(x)$, we can securely calculate the indices $\mathbf{b}_1, \ldots, \mathbf{b}_\ell$ that are used as input to the protocol $\mathsf{SPIR}^{\delta}_{m}$. Recall that $\mathbf{b}_k = (b_{k,1}, \ldots, b_{k,m})$

is the encryption of an m-length bit-string of Hamming weight 1, selecting the appropriate item from each m sized bucket at step k. If we consider the buckets to be arranged consecutively (the first m elements in the first bucket, and so on) then $b_{k,j} = E(c_{k,j})$ where

$$c_{k,j} = (x \bmod m^k \stackrel{?}{=} (j-1)m^{k-1} + \sum_{h=1}^{k-1}\sum_{i=1}^{m}(i-1)c_{h,i}m^{h-1})$$

Thus, $\mathbf{b}_k$ can be calculated using MOD, EEQTEST, and the vectors $\mathbf{b}_j$ for $j < k$ calculated in earlier rounds. Each round, this procedure takes polylog work with polylog communication complexity. The procedure ends after $\ell = O(\sqrt{\log n})$ rounds.

Full Privacy Preserving Implementation. The secure implementation of our new variant of Gale-Shapley is assembled using the protocols indicated above, according to the algorithm below.

Input submission and Initialization
For $k = 1$ to $2n^2$:

1. Select the single free bid W_i from $\mathcal{F}_k$.
2. Open W_i to recover $E(j), E(s_{j,i})$
3. Form the engaged bid $\langle W_i, E(j), E(s_{j,i})\rangle$
4. Find the conflicting engaged bid, $\langle W_{i'}, Ej, E(s_{j,i'})\rangle$
5. Mix these two engaged bids
6. Resolve the conflict to find the "winner" and "loser"
7. Break the engagement for the loser and add this free bid to $\mathcal{F}_{k+1}$
8. Add the winner to $\mathcal{E}_k$
9. Mix the engaged bids
10. Let $\mathcal{E}_{k+1} = \mathcal{E}_k$

Announce stable marriage

Complexity Analysis. As in Golle's, the communication complexity here is dominated by the re-encryption mix networks run in steps 4, 5, and 9 — specifically, the mixnets run in steps 4 and 9. These mixnets runs on $O(n)$ ciphertexts each round. The total communication complexity is $O(tn^3)$. Accessing the database, however, dominates the computational cost, when $t \leq n$. Each step k, the database access takes $O(n^2\sqrt{\log n})$ work. The total computational complexity is $O(n^4\sqrt{\log n})$. The round complexity is $O(n^2\text{polylog}(n))$.

Claim. The algorithm of Section 4.2 is a private stable matching protocol, assuming Paillier encryption is semantically secure and the underlying re-encryption mix network is private.

We note the above claim can be proven by a hybrid argument similar to that of the proof of Proposition 3 in [12], or using the composition theorems mentioned in Section 2.2. Due to length constraints (and the lack of novelty in applying either of these proof techniques), we omit a full proof.

5 An Efficient Private Stable Matching Protocol for $t = 2$ MAs

In this section, we take a closer look at the case where there are only two Matching Authorities (MAs). We design a secure protocol for this case with $O(n^2\text{polylog}(n))$ communication complexity. This is a factor of n more efficient than our protocols for the general case. We generalize this protocol for the setting with multiple pairs of MAs in Appendix B of the full version of this paper [8].

We base our secure implementation on our variant of Gale-Shapley from Section 3.3. To do so, we use rather different techniques from those we used in the general case. As before, each participant sends shares of his/her input to the two MAs. The rest of the protocol is performed by the two MAs without help from the participants. The final matching is then revealed to the participants. Before we proceed with the details of the protocol, let us define the data structures that are shared by the MAs. For simplicity, in what follows we label men and women using only their index.

$A[i][j] = a_{i,j}$,	the identity of the woman ranked $j-1$ by man A_i.
$B[j][i] = s_{j,i} + 1$,	where $s_{j,i} \in [0, n-1]$ is the rank given to man A_i by woman B_j.
$P[i] = \rho_i + 1$,	where $\rho_i \in [0, n-1]$ is the rank of the woman to whom man A_i will propose next.
$E[j] \in \{1, \ldots, n\}$,	the identity of the man engaged to woman B_j.

Using the above data structures, we can rewrite our variant of the Gale-Shapley algorithm (from Section 3.3), after the initialization stage, as follows.

For $k = 1$ to $2n^2$

1. Remove i from $\mathcal{F}_k$
2. Let $p = P[i]$
3. Let $j = A[i][p]$ (the index of the woman to whom A_i proposes)
4. Let $i' = E[j]$ (the index of the man currently engaged to her)
5. Let $s_{j,i} = B[j][i]$ and $s_{j,i'} = B[j][i']$ (her rankings for A_i and $A_{i'}$)
6. If $s_{j,i'} > s_{j,i}$, then swap the labels i, i'
7. $E[j] \leftarrow i'$ (store the "winner" as her husband)
8. $p' = P[i]$
9. $P[i] \leftarrow p' + 1$ (increment the "loser")
10. $\mathcal{F}_{k+1} = \{i\}$ (free the "loser")

At the end of the protocol, $E[j]$ stores the index of the man to whom woman B_j is married. The MAs can privately (or publicly) announce to each participant shares of his/her partner. Now, we explain how to implement the above algorithm securely. Note that all the data structures and intermediate values in the algorithm are shared between the two MAs. To be compatible with the private table access primitives we use, we employ a simple XOR sharing scheme: to share a k-bit integer a between the MAs, a participant sends a random k-bit r to one MA and sends $a \oplus r$ to the other MA.

In steps 2–5 and 7–9, MAs need to privately read and write to a table. We can use the techniques of Naor and Nissim [21] to implement these steps securely (see Section 2.3 for more detail), with $O(\text{polylog}(n))$ communication. We can do step 6 (compare two

integers and potentially swapping them) and step 9 (computing shares of $\rho + 1$ from shares of ρ) by switching to Yao's garbled circuit protocol and then switching back to the initial setting (see Section 2.3 for more detail). The circuit for performing such computations is of size $O(\text{polylog}(n))$. This leads to a protocol with $O(n^2\text{polylog}(n))$ communication between the MAs.

According to the composition theorems with respect to passive adversaries (see Section 2.2), the above protocol is privacy-preserving as long as the underlying subprotocols (private table read/write protocols and Yao's garbled circuit protocol) are secure against passive adversaries.

References

1. Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *CRYPTO '02*, pages 417–432, 2002.
2. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *ACM Symposium on Theory of Computing*, pages 503–513, 1990.
3. Ran Canetti. Security and composition of multiparty cryptographic protocols. In *Journal of Cryptology*, volume 13, pages 143–202, 2000.
4. Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In *CRYPTO '01*, pages 119–136, 2001.
5. Ivan Damgård, Matthias Fitzi, Jesper Buus Nielsen, and Tomas Toft. How to split a shared secret into shared bits in constant-round. Cryptology ePrint Archive, Report 2005/140, 2005.
6. Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136, 2001.
7. Pierre-Alain Fouque, G Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Financial Crypto*, 2000.
8. Matthew Franklin, Mark Gondree, and Payman Mohassel. Improved efficiency for private stable matching. Cryptology ePrint Archive, Report 2006/332, 2006.
9. David Gale and Lloyd Stowell Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
10. Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.
11. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *ACM Symposium on Theory of Computing*, pages 218–229, 1987.
12. Philippe Golle. A private stable matching algorithm. In *Financial Crypto*, 2006.
13. Philippe Golle and Ari Juels. Parallel mixing. In *ACM Computer and Communications Security*, pages 220–226, 2004.
14. Dan Gusfield and Robert Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.
15. Markus Jakobsson, Ari Juels, and Ron Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proc. of USENIX'02*, pages 339–353, 2002.
16. Markus Jakobsson and Claus Peter Schnorr. Efficient oblivious proofs of correct exponentiation. In *Communications and Multimedia Security*, pages 71–86, 1999.
17. Eike Kiltz. Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation. Cryptology ePrint Archive, Report 2005/066, 2005.
18. Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Foundations of Computer Science*, pages 364–373, 1997.

19. Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004.
20. Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *ASIACRYPT 2003*, pages 416–433, 2003.
21. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *ACM Symposium on Theory of Computing*, pages 590–599, 2001.
22. C. Andrew Neff. A verifiable secret shuffe and its application to e-voting. In *ACM Computer and Communications Security*, pages 116–125, 2001.
23. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, pages 223–238, 1999.
24. Julien P. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *ASIACRYPT'98*, pages 357–371, 1998.
25. Andrew C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science*, pages 162–167, 1986.

A A Case Where [12] Requires More Than n Iterations

Here we give an example input for which Golle's variant of Gale-Shapley requires $\Omega(n^2)$ iterations of the main loop to reach a stable matching. Consider the following preference lists for the real men and women (the preference lists of the n fake men are as indicated in [12]):

A_1	$[1, 2, \ldots, n-1, n]$	B_1	$[2, 3, \ldots, n, 1]$
A_2	$[2, 3, \ldots, 1, n]$	B_2	$[3, 4, \ldots, 1, 2]$
	$\ldots$		$\ldots$
A_{n-1}	$[n-1, 1, \ldots, n-2, n]$	B_{n-1}	$[n, 1, \ldots, n-2, n-1]$
A_n	$[1, 2, \ldots, n-1, n]$	B_n	$[1, 2, \ldots, n-1, n]$

After the first iteration of the main loop, $n-1$ real men are engaged and 1 real man remains free. This implies that $\mathcal{F}_2$ will include $n-1$ fake men. In the next n^2-2n+2 iterations, one real man will get engaged and another will become free. In other words, for $2 \leq k \leq n^2-2n+2$, only one real man will propose in each iteration and all the other proposals are made by fake men. Thus, $\Omega(n^2)$ iterations of the main loop is necessary to reach a stable matching.

Compact E-Cash from Bounded Accumulator

Man Ho Au, Qianhong Wu, Willy Susilo, and Yi Mu

Center for Information Security Research
School of Information Technology and Computer Science
University of Wollongong, Wollongong 2522, Australia
{mhaa456,qhw,wsusilo,ymu}@uow.edu.au

Abstract. Known compact e-cash schemes are constructed from signature schemes with efficient protocols and verifiable random functions. In this paper, we introduce a different approach. We construct compact e-cash schemes from bounded accumulators. A bounded accumulator is an accumulator with a limit on the number of accumulated values. We show a generic construction of compact e-cash schemes from bounded accumulators and signature schemes with certain properties and instantiate it using an existing pairing-based accumulator and a new signature scheme. Our scheme revokes the secret key of the double-spender directly and thus supports more efficient coin tracing. The new signature scheme has an interesting property that is has the message space of a cyclic group $\mathbb{G}_1$ equipped with a bilinear pairing, with efficient protocol to show possession of a signature without revealing the signature nor the message. We show that the new scheme is secure in the generic group model. The new signature scheme may be of independent interest.

Keywords: compact e-cash, bounded accumulator, bilinear pairings.

1 Introduction

Electronic cash(e-cash), introduced by Chaum[8], is one of the useful ways to pay electronically. It could play an important role in the realization of fully online commerce. A practical electronic cash system should be *secure*, *offline* and *anonymous*. Depending on the implementation, it may or may not require some proprietary hardware.

An *e-cash* system consists of three parties (the bank $\mathcal{B}$, the user $\mathcal{U}$ and the shop $\mathcal{S}$) and four main procedures (account establishment, withdrawal, payment and deposit). The user $\mathcal{U}$ first performs an account establishment protocol with the bank $\mathcal{B}$. The currency circulating around is quantized as coins. $\mathcal{U}$ obtains a coin by performing a withdrawal protocol with $\mathcal{B}$ and spends the coin by participating in a payment protocol with $\mathcal{S}$. To deposit a coin, $\mathcal{S}$ performs a deposit protocol with $\mathcal{B}$.

Security of e-cash refers to the fact that only the bank $\mathcal{B}$ can produce a coin and for offline schemes, users who double-spent should be identified. The problem of double-spending occurs in the electronic world due to easy duplication of digital coins. Additionally, honest spenders cannot be slandered to have double

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 178–195, 2007.

spent (*exculpability*), and when the shops deposit the money from the payee, the bank should not be able to trace who the actual spender is (*anonymity).* Many e-cash systems that provide the function of identifying double-spenders have been proposed, but most of them rely on a trusted third party (TTP) to *revoke* the anonymity so as to identify the double-spenders [4,12,7]. While the TTP cannot slander an honest user, its existence in fact implies that even honest users are not anonymous.

It is desirable to have *coin traceability* of double-spender such that all coins of the cheating user can be traced. Certain information can be put in a blacklist so that coins from the double-spenders can be recognized when it is spent.

High *efficiency* is also of key importance for practical *e-cash* systems. For efficiency, we look at: (1) the time and bandwidth needed for the withdrawal, payment and deposit protocols; (2) the size of an electronic coin; and (3) the size of the bank's database.

Camenisch, Hohenberger and Lysyanskaya [5] proposed a secure offline anonymous e-cash scheme (which we shall refer to as CHL scheme from now on) which is compact to address the efficiency issue. In their scheme, a wallet containing k coins can be withdrawn and stored in complexity $O(\lambda+\log(k))$ for a security parameter λ, where each coin can be spent unlinkably with complexity $O(\lambda+\log(k))$ as well.

CHL's Compact E-Cash. There are two versions of CHL's scheme with *coin traceability.* We describe the one with the XDH assumption since it is simpler and much more efficient than the other. When a user withdraws a wallet, it obtains a signature σ from the bank on the commitment of user's secret key and two secret random numbers, s and t. To spend a coin, the user proves to the merchant that it possesses such a signature σ, and uses s to generate a link tag. By using a verifiable random function on s and counter i, the user generates a link tag S_i (called serial number). The user also generates a blinding value B_i, using the same method, from t and counter i. For the counter i running from 0 to $2^\ell-1$, the wallet can be spent 2^ℓ times unlinkably. The blinding value is used to compute a value $T_i = g^u B_i^R$, where g^u is the secret key of the user and R is a random challenge provided by the merchant. T_i is called a double-spending equation. Should a user double-spent any of his coins, two double-spending equations can be used to reveal the secret key of the user. For full coin tracing, user verifiably encrypt certain tracing information (s, t) under his own public key. Once his secret key is revealed due to double-spending, the bank can decrypt the tracing information and trace all the spendings of the double-spender. Due to the nature of the revocation mechanism, the secret key is limited to the form of g^u. Thus, CHL scheme employs a rather inefficient cut-and-choose verifiable encryption based on the bilinear encryption[2].

Overview of Our Scheme. We present an intuition on how our scheme is constructed. Let g, h be generators of a cyclic group of prime order p. Let $y = g^x$ for some $x \in \mathbb{Z}_p$. It has been well known that a proof-of-knowledge of the discrete logarithm of y can be done, with the following 3 moves.

- Commitment. The prover chooses a random $r \in_R \mathbb{Z}_p$, compute $T = g^r$ and sends it to the verifier.
- Challenge. The verifier randomly generates a challenge $c \in_R \mathbb{Z}_p$ and sends it back to the prover.
- Response. The prover computes a response $z = r - cx$ and sends it to the verifier.

The verifier checks if $T = y^c g^z$ and if it holds, it is convinced that the prover indeed knows the discrete logarithm of y to the base g. Should the prover uses the same commitment and receives different challenges to produce different responses, x can be computed efficiently. The existence of such algorithm, known as knowledge extractor, is needed to prove that the proof system is a proof of knowledge[10]. Such technique is employed in Brands' E-cash [3] to reveal the identity of the double-spender.

Now we try to use the same rationale to construct our compact e-cash scheme as follows. Each user, equipped with key pairs (x, g^x) first generates k random numbers r_i for $i = 1 \cdots k$ and uses an accumulator[6,13] to accumulate the set of random numbers $\{r_i\}$ into an accumulated value V and present it to the bank. The bank then signs V, g^x together. When the user wishes to spend a coin, he/she uses one of the random number r_i and computes the a serial number $S = h^{r_i}$, a masked commitment $T = h^{r_i} g^{k_1}$ and a masked public key $Y = h^x g^{k_2}$, where k_1, k_2 are the some random values. The user, submit S, T, Y, and proves to the merchant, in zero knowledge manner, that he is in possession of the bank's signature on V, g^x and S, T, Y are correctly formed. Finally, the user uses T as the commitment to prove the knowledge of x, k_2. Should a user attempt to spend more than k coins, he will have to use the same S and be identified. Next his secret key x shall be revealed from the component T, Y.

That is almost our final solution. To achieve full coin tracing, we borrow the idea from traceable signatures[11]. Instead of having the group manager storing the tracing information, we require the user to choose the tracing information and verifiably encrypt it under the public key of himself. When his private key is revealed due to double-spending, all his spendings can be traced. Two problems remain, the first one being how can the bank ensure that the user only accumulates k values in V but not more. The second problem is that there is no existing signature with efficient protocols which allows signing on the space of the accumulator, usually a group element itself, directly. We solve the first problem by making the observation that the accumulator due to Nguyen[13] in fact impose an upper bound on the number of values to be accumulated. We solve the second problem by introducing a new signature scheme, which could be of independent interest.

Our Contributions. Specifically, we make the following contributions

- We present generic construction for compact e-cash scheme from bounded accumulator.
- We provide efficient instantiation.
- We propose a new signature scheme with efficient protocols, which may be of independent interest.

Organization. We discuss related works and technical preliminary in the next section. A security model is shown in Section 3. The generic construction is shown in Section 4, accompanied by security analysis. Next we present the building blocks of our instantiations in Section 5. We show our instantiation in Section 6 and concluded with some remarks in Section 7.

2 Preliminaries

2.1 Notations

Let e be a bilinear map such that $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$.

- $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic multiplicative groups of prime order p.
- each element of $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_3$ has unique binary representation.
- g, h are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.
- (Bilinear) $\forall x \in \mathbb{G}_1$, $y \in \mathbb{G}_2$ and a, $b \in \mathbb{Z}_p$, $\hat{e}(x^a, y^b) = \hat{e}(x, y)^{ab}$.
- (Non-degenerate)$\hat{e}(g, h) \neq 1$.

$\mathbb{G}_1$ and $\mathbb{G}_2$ can be same or different groups. We say that two groups ($\mathbb{G}_1$, $\mathbb{G}_2$) are a bilinear group pair if the group action in $\mathbb{G}_1$, $\mathbb{G}_2$ and the bilinear mapping e are all efficiently computable.

2.2 Mathematical Assumptions

Definition 1 (Decisional Diffie-Hellman). *The Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ is defined as follow: On input a quadruple $(g, g^a, g^b, g^c) \in \mathbb{G}^4$, output 1 if $c = ab$ and 0 otherwise. We say that the (t, ϵ)-DDH assumption holds in $\mathbb{G}$ if no t-time algorithm has advantage at least ϵ over random guessing in solving the DDH problem in $\mathbb{G}$.*

Definition 2 (Symmetric External Diffie-Hellman[1]). *The Symmetric External Diffie-Hellman (SXDH) Assumption state that the DDH problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$ of a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$. It implies that there is no efficiently computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ or vice versa.*

2.3 Bounded Accumulator

We introduce a new notion, bounded accumulator $\mathcal{BA}$ as follows. A bounded accumulator is a dynamic accumulator[6,9] with the constraint that at most k, called the bound of the accumulator, values could be accumulated into the accumulator.

3 Syntax

A compact e-cash with full coin tracing is a tuple (BankSetup, UserSetup, WithdrawalProtocol, SpendProtocol, DepositProtocol, RevokeDoubleSpender, Trace) of seven polynomial time algorithms/protocols between three entities, namely Bank, Merchant and User. The following enumerates the syntax.

- BankSetup. On input an unary string 1^λ, where λ is the security parameter, the algorithm outputs the bank's master secret bsk and the public parameter bpk.
- UserSetup. On input bpk, the algorithm outputs a key pair (pk, sk). Since merchants are a subset of users, they may use this algorithm to obtain keys as well.
- WithdrawalProtocol. The user with input (pk, sk) withdraws a wallet w of k coins from the bank. The bank's input is the master secret bsk. After executing the protocol, the user obtains a wallet w while the bank (possibly) retains certain information τ_w, called the trace information.
- SpendProtocol. This is the protocol when the user spends a single coin to a merchant. The user input is w and the merchant's identity. After the protocol, the merchant obtains a transcript including a proof of validity π of a coin from the wallet, and possibly some auxiliary information aux, and outputs 0/1, depending whether the payment is accepted. The user's output is an updated wallet w'.
- DepositProtocol. In a deposit protocol, the merchant submits (π, aux) to the bank for deposit. The bank outputs 0/1, indicating whether the deposit is accepted. It is required whenever an honest merchant obtains (π, aux) by running any of the spend protocols with some user, there is a guarantee that this transaction will be accepted by the bank. The bank adds (π, aux) to the database of spent coins.
- RevokeDoubleSpender. Whenever a user double spent, this algorithm allows the bank to identify the double spender. Formally, on input two spending protocol transcripts involving the same coin, the algorithm outputs the public key pk of the double-spender.
- Trace. On input two spending protocol transcripts of a double-spender, the algorithm outputs a certain information tr which allows everyone to identify all the spendings of the double-spender.

3.1 Security Notions

We first provide an informal description of the security requirements. A *secure* compact e-cash scheme should possess *correctness*, *balance*, *CorrectTracingOfDoubleSpender*, *anonymity* and *exculpability*, introduced as follows.

- *Correctness.* If an honest user runs WithdrawalProtocol with an honest bank and runs any of the spend protocols with an honest merchant, the merchant accepts the payment. The merchant later runs Deposit with the bank, which will accept the transaction.
- *Balance.* This is the most important requirement from the bank's point of view. Roughly speaking, *balance* means that no collusion of users and merchants together can deposit more than they withdraw without being identified. More precisely, we require that collusion of users and merchants, having run the withdrawal protocol for n times, cannot deposit more than nk coins back to the bank without being identified, that is, if they do deposit $nk + 1$

coins, at least one of the colluders must be identified. A related notion is revocability, which means identity of the double-spender must be revoked. It is straightforward to see that revocability is implied by the definition of *balance*.
- *CorrectTracingOfDoubleSpender.* It means that should a user be a double-spender, all his other spendings should be properly identified.
- *Anonymity.* It is required that no collusion of users, merchants and the bank can ever learn the spending habit of an honest user.
- *Exculpability.* It is required that an honest user cannot be proven to have double-spent, even all other users, merchants and the bank colludes.

From our definition, it can be seen that it is the bank's responsibility to identify the double-spender. The rationale behind is that a user can always spend the same coin to different merchants in an offline e-cash system and the merchant have no way to detect such double-spending.

Next we are going to formally define the security model. The capability of an adversary $\mathcal{A}$ is modeled as oracles.

- Withdrawal Oracle: $\mathcal{A}$ presents a public key pk and engages in the WithdrawalProtocol as user and obtains a wallet. The oracle stores pk in a set $\mathbb{X}_A$.
- Spend Oracle: $\mathcal{A}$ now acts as a merchant and request users to spend coins with it.
- Hash Oracle: $\mathcal{A}$ can ask for the values of the hash functions for any input.

We require that the answers from the oracles are indistinguishable from the view as perceived by an adversary in real world attack.

Definition 3 (Game Balance)

- (Initialization Phase.) *The challenger $\mathcal{C}$ takes a sufficiently large security parameter λ and runs* BankSetup *to generate bpk and also a master secret key bsk. $\mathcal{C}$ keeps bsk to itself and sends bpk to $\mathcal{A}$.*
- (Probing Phase.) *The adversary $\mathcal{A}$ can perform a polynomially bounded number of queries to the oracles in an adaptive manner.*
- (End Game Phase.) *Let q_w be the number of queries to the Withdrawal Oracle and q_s be the number of queries to the Spend Oracle. $\mathcal{A}$ wins the game if it can run $kq_w + q_s + 1$ deposits to $\mathcal{C}$ such that, on input any two of these $kq_w + q_s + 1$ deposits transcript,the* RevokeDoubleSpender *algorithm does not output any of the public keys presented during the Withdrawal Oracle query.*

The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins.

Definition 4 (Game CorrectTracing). *The game to be played is the same as game balance, except we are more generous as to what constitutes the adversary's success. Basically the $\mathcal{A}$ wins if after a double spending has been detected, it could produce another spending, under the same public key, that could not be traced.*

Definition 5 (Game Anonymity)

- (Initialization Phase.) *The challenger $\mathcal{C}$ gives a sufficiently large security parameter λ to $\mathcal{A}$. $\mathcal{A}$ then generates bpk and bsk. $\mathcal{A}$ gives bpk to $\mathcal{C}$. Since $\mathcal{A}$ is in possession of bsk, only Hash oracle query is allowed in Game Anonymity.*
- (Challenge Phase.) *$\mathcal{C}$ then chooses two public keys PK and PK' and presents them to $\mathcal{A}$. $\mathcal{C}$ runs the* WithdrawalProtocol *with $\mathcal{A}$ acting as bank to obtain several wallets $w_0, \cdots, w_t$ and $w'_0, \cdots, w'_t$ on behalf of the two public keys. $\mathcal{A}$ then acts as merchant and ask for spending from $\mathcal{C}$. $\mathcal{A}$ is allowed to specify which wallet $\mathcal{C}$ uses, with the restriction that it cannot ask $\mathcal{C}$ to over-spend any of the wallets. Finally, $\mathcal{C}$ randomly chooses one wallet w from user PK and one wallet w' from user PK' from the set of wallets that are legal for the challenge, flip a fair coin to decide to use w or w' for the challenge spending.*
- (End Game Phase.) *The adversary $\mathcal{A}$ decides which public key $\mathcal{C}$ uses.*

$\mathcal{A}$ wins the above game if it guesses correctly. The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins minus $\frac{1}{2}$.

Definition 6 (Game Exculpability)

- (Initialization Phase.) *The challenger $\mathcal{C}$ gives a sufficiently large security parameter λ to $\mathcal{A}$. $\mathcal{A}$ then generates bpk and bsk. $\mathcal{A}$ gives bpk to $\mathcal{C}$. Since $\mathcal{A}$ is in possession of bsk, only Hash oracle query is allowed in Game Exculpability.*
- (Challenge Phase.) *$\mathcal{C}$ runs the* WithdrawalProtocol *for q_j times with $\mathcal{A}$ acting as bank to obtain wallets $w_1, \cdots, w_{q_j}$. $\mathcal{A}$ then acts as merchant and asks for spending from $\mathcal{C}$. $\mathcal{A}$ is allowed to specify which wallet $\mathcal{C}$ uses, with the restriction that it cannot ask $\mathcal{C}$ to over-spend any of the wallets. $\mathcal{A}$ can also ask to corrupt any of the user in the above withdrawal protocol. A corrupted user needs to surrender its private key as well as the wallet to $\mathcal{A}$.*
- (End Game Phase.) *$\mathcal{A}$ runs two deposit protocols with $\mathcal{C}$. $\mathcal{A}$ wins the game if* RevokeDoubleSpender *on this two deposit protocols points to a user in* any *of the withdrawal protocol during initialization and that user has not been corrupted.*

The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins.

A compact e-cash with full tracing is *secure* if no PPT adversary can win in Game Balance, Game CorrectTracing, Game Anonymity and Game Exculpability with non-negligible advantage.

4 Generic Construction

We present a generic construction of compact e-cash scheme from bounded accumulator. For better presentation, we first describe the basic system. Next, we extend the basic system to support full tracing.

4.1 Basic System

BankSetup. Let $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme. Let $\mathcal{BA}$ be a bounded accumulator with upper bound k. The bank runs KeyGen and obtains (sk, pk). The bank publishes $pk, \mathcal{BA}$ as the public key. It keeps sk as the private key. Each user is in possession of a DL type key pairs (x, u^x).

Withdrawal. To withdraw, the user generates k random numbers s_i for $i = 1, \cdots, k$. He accumulates the k s_i's to form an accumulated value v and obtain k witnesses w_i. The user then obtains a signature $\sigma_x = \mathsf{Sign}(v, \mathrm{Commit}(x))$ from the bank. The user keeps $(\sigma_x, \{s_i\}, \{w_i\}, x)$ as its wallet secret.

Spend Protocol. For payment, the user and the merchant with identity $I \in \{0,1\}^*$ first agree on the transaction information info. The user then chooses one of the unused random number in his wallet and compute the following quantities, $S = g^{s_i}$, $T = g^{r_1} h^{s_i}$, $Y = g^{r_2} h^x$. The user then generates an non-interactive zero-knowledge proof-of-knowledge (signature of knowledge) π_1 of the following.

- $\mathsf{Verify}(\sigma_x, v, \mathrm{Commit}(x)) = 1$
- s_i is in v (using w_i as witness)
- S, T, Y are correctly formed (with respect to the $v, \mathrm{Commit}(x)$)

Finally, the user computes a signature of knowledge π_2 on the representation of Y, using T as the commitment. This involves computing $(z_r = r_1 - cr_2, z_x = s_i - cx)$ such that $T = Y^c g^{z_r} h^{z_x}$, where c is the challenge used in the signature of knowledge. In both signature of knowledge π_1, π_2, the message to be signed is $(I||\text{info})$, where $||$ denote the concatenation operator.

The merchant accepts the payment if both π_1 and π_2 are valid.

Deposit. To deposit, the merchant simply gives the bank the whole communication transcript during the spend protocol. The bank verifies the transcript exactly as the merchant did. In addition, the bank has to verify that I is indeed the identity of the merchant and (info, I) is not used before by that merchant. This is to prevent colluding users and merchants from submitting a double spent coin (which have identical transcripts). It also prevents malicious merchant from eavesdropping honest transaction and depositing it (in that case, identity of the malicious merchant does not match with I). In case the check is successful, the bank stores (S, c, z_x) to the database.

RevokeDoubleSpender. When a new spending transcript is received, the bank checks if S exists in the database. If yes, then it is a double-spent coin. The bank identifies the double-spender by compute $x = \frac{z_x - z'_x}{c' - c}$. Output the identity of the double-spender as u^x.

4.2 Compact E-Cash with Full Coin Tracing

To extend our system to support full tracing, we make use of the idea from event-oriented linkable ring signatures. We highlight the differences as follow.

Withdrawal. During withdrawal, the user obtains a signature $\sigma_x = \mathsf{Sign}(v, \mathrm{Commit}(x, tr))$ for the additional random number tr. The user also needs to verifiably encrypts tr under his own public key u^x. Since the encryptor knows the private key x, in contrast with the normal case for verifiable encryption where the encryptor does not have the private key of the decryptor, we can have a very efficient implementation, as shown in the implementation. The bank keeps the encryption of tr in the database.

SpendProtocols. The user needs to compute an additional tracing tag equals to $tag = H(I||\mathrm{info})^{tr}$ for some cryptographic hash function H whose range is a cyclic group where the DDH problem is hard and the modified π_1 is shown below.

- $\mathsf{Verify}(\sigma_x, v, \mathrm{Commit}(x, tr)) = 1$
- s_i is in v (using w_i as witness)
- S, T, Y, tag are correctly formed (with respect to the $v, \mathrm{Commit}(x, tr)$)

FullTracing (of the double-spender). As the secret key x of the double-spender is computed, the bank can decrypt tr. Once tr is published, all shops can check if $tag = H(I||\mathrm{info})^{tr}$ and identify the spendings of the double-spender. Note that this tracing is regardless of wether the coin have been spent or not. All the money withdrawn under the public key of the double-spender can be traced.

4.3 Security of the Generic Construction

Intuitively, the generic construction is secure, suppose the underlying signature scheme is existentially unforgeable against chosen message attack, the accumulator is bounded, collision resistant and with one-way domain, the protocols are honest-verifier zero-knowledge with special soundness, together with an extra condition stated as follows.

Condition `Acc`: Let $\{x_1, \cdots, x_n\}$ be a set of random values in the domain of the accumulator. Let $< g >$ be a cyclic group whose order is also the domain of the accumulator. Given $g^{x_1}, g^{x_2}, \cdots, g^{x_n}$ and v, it is difficult to decide if v is the accumulation of $\{x_i\}$.

Proof of the claim shall appear in the full version of the paper.

5 Building Blocks of Our Instantiation

In theory, zero-knowledge proof-of-knowledge exists for any NP relations. However, efficient zero-knowledge proof-of-knowledge protocols may not exist for all signature schemes. In particular, our generic construction requires a signature scheme which can sign on elements in $\mathbb{G}_1$, together with efficient proof-of-knowledge protocols for showing possession of a signature without revealing the message. Unfortunately, such scheme does not exist in the literature, to the best of the authors' knowledge. We propose the following signature scheme, accompanied by the security analysis in the generic group model, followed by an extension which allows the signer to sign an element in $\mathbb{G}_1$ together with a block of n messages in $\mathbb{Z}_p^n$.

5.1 Special Signature

KeyGen. On input 1^λ, where λ is a security parameter, output $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ of order p where p is a λ-bit prime. Also output a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$, g_1, g_2, x, y, $X = g_1^x$, $Y = g_2^y$, $A = e(X, g_2)$ such that g_1, g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. The public key is $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e, g_1, g_2, A, Y)$ and the secret key is (X, y).

Sign. For any message M in $\mathbb{G}_1$, randomly choose $r \leftarrow \mathbb{Z}_p^*$, compute $a_1 = XM^r, a_2 = g_1^{\frac{1}{y+r}}, a_3 = g_2^r$. The resulting signature is (a_1, a_2, a_3).

Verify. Check that $e(a_1, g_2) = Ae(M, a_3)$ and $e(a_2, a_3 Y) = e(g_1, g_2)$. Output 1 if both equations hold. Else, output 0.

5.2 Security Analysis of the Signature Scheme

We derive an upper bound on the success probability of an adversary $\mathcal{A}$ that could existentially forge a signature under adaptively chosen message attack in the generic group model against our signature scheme. Under the SXDH assumption, there is no efficient distortion maps between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Let $\Upsilon = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, g_1, g_2, e) \leftarrow \mathtt{PairingGen}(1^\lambda)$, $x, y \leftarrow \mathbb{Z}_p^*$, $X = g_1^x$,$A = e(X, g_2)$, $Y = g_2^y$. Let $\mathcal{O}_X(\cdot)$ be an oracle that takes as input $M \in \mathbb{G}_1$, and outputs a tuple $(XM^r, g_1^{\frac{1}{r+y}}, g_2^r)$ for a random $r \in \mathbb{Z}_p^*$. Let $\mathbb{M}$ denote the set of M that has been queried to $\mathcal{O}_X(\cdot)$. Then for any PPT adversaries $\mathcal{A}(\cdot)$ in the generic group model, making a total of τ queries to the oracles computing the group action in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$, the oracle computing the bilinear pairing e and the oracle $\mathcal{O}_X(\cdot)$, if $x \in \mathbb{Z}_p^*$ is chosen at random, then the success probability of $\mathcal{A}$ is

$$\Pr\left[\begin{array}{l|l} x, y \leftarrow \mathbb{Z}_p^* & (M', \sigma_1, \sigma_2, \sigma_3) \leftarrow \mathcal{A}^{\mathcal{O}_X(\cdot)}(g_1, g_2, A, Y) \\ Y = g_2^y & \wedge M' \notin \mathbb{M} \wedge M' \in \mathbb{G}_1 \wedge \sigma_1 = XM'^r \\ A = e(g_1^x, g_2) & \wedge \sigma_2 = g_1^{\frac{1}{r+y}} \wedge \sigma_3 = g_2^r \end{array}\right] \leq O(\tau^3/p)$$

Proof. Consider an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ in the following game.

$\mathcal{B}$ maintains three lists of pairs $L_1 = \{(F_{1,i}, \xi_{1,i}) : i = 0, \cdots, \tau_1 - 1\}$, $L_2 = \{(F_{2,i}, \xi_{2,i}) : i = 0, \cdots, \tau_2 - 1\}$, $L_3 = \{(F_{3,i}, \xi_{3,i}) : i = 0, \cdots, \tau_3 - 1\}$, such that, at step τ in the game, we have $\tau_1 + \tau_2 + \tau_3 = \tau + 6$. The $\xi_{1,i}$, $\xi_{2,j}$, and $\xi_{3,k}$ are set to unique random strings in $\{0,1\}^*$. $\mathcal{B}$ starts the game at step $\tau = 0$ with $\tau_1 = 1$, $\tau_2 = 2$, and $\tau_3 = 3$. These correspond to the multivariate polynomial functions $F_{1,0} = F_{2,0} = F_{3,0} = 1$, $F_{3,1} = x$, $F_{2,1} = F_{3,2} = y$.

$\mathcal{B}$ begins the game with $\mathcal{A}$ by providing it with the 3 strings $\xi_{1,0}$, $\xi_{2,0}$, $\xi_{2,1}$, $\xi_{3,1}$. Now, we describe the oracles that $\mathcal{A}$ may query.

Group action: $\mathcal{A}$ inputs two group elements $\xi_{1,i}$ and $\xi_{1,j}$, where $0 \le i, j < \tau_1$, and a request to multiply/divide. $\mathcal{B}$ sets $F_{1,\tau_1} \leftarrow F_{1,i} \pm F_{1,j}$. If $F_{1,\tau_1} = F_{1,\ell}$ for some $\ell \in \{0, \cdots, \tau_1 - 1\}$, then $\mathcal{B}$ sets $\xi_{1,\tau_1} = \xi_{1,\ell}$; otherwise, it sets ξ_{1,τ_1} to a random string in $\{0,1\}^* \setminus \{\xi_{1,0}, \cdots, \xi_{1,\tau_1-1}\}$. Finally, $\mathcal{B}$ returns ξ_{1,τ_1} to $\mathcal{A}$, adds

$(F_{1,\tau_1}, \xi_{1,\tau_1})$ to L_1, and increments τ_1 by one. Group actions for $\mathbb{G}_2$ and $\mathbb{G}_3$ are handled in the same way.

Pairing: $\mathcal{A}$ inputs two group elements $\xi_{1,i}$ and $\xi_{1,j}$, where $0 \leq i < \tau_1$ and $0 \leq j < \tau_2$. $\mathcal{B}$ sets $F_{3,\tau_3} \leftarrow F_{1,i} \cdot F_{2,j}$. If $F_{3,\tau_3} = F_{3,\ell}$ for some $\ell \in \{0, \cdots, \tau_1 - 1\}$, then $\mathcal{B}$ sets $\xi_{3,\tau_3} = \xi_{3,\ell}$; otherwise, it sets ξ_{3,τ_3} to a random string in $\{0,1\}^* \setminus \{\xi_{3,0}, \cdots, \xi_{3,\tau_3-1}\}$. Finally, $\mathcal{B}$ returns ξ_{3,τ_3} to $\mathcal{A}$, adds $(F_{3,\tau_3}, \xi_{3,\tau_3})$ to L_3, and increments τ_3 by one.

We do not allow efficient distortion maps between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Oracle $\mathcal{O}_X(\cdot)$: $\mathcal{A}$ inputs $\xi_{1,m}$, where $0 \leq m < \tau_1$, followed by $\mathcal{B}$ choosing a new variable s_{τ_s} and setting $F_{2,\tau_2} \leftarrow F_{2,0} \cdot s_{\tau_s}$, $F_{1,\tau_1} \leftarrow F_{1,0}(x + ms_{\tau_s})$, $F_{1,\tau_1+1} \leftarrow F_{1,0}\frac{1}{s_{\tau_s}+y}$. If $F_{2,\tau_2} = F_{2,\ell}$ for some $\ell \in \{0, \cdots, \tau_2 - 1\}$, then $\mathcal{B}$ sets $\xi_{2,\tau_2} = \xi_{2,\ell}$; otherwise, it sets ξ_{2,τ_2} to a random string in $\{0,1\}^* \setminus \{\xi_{2,0}, \cdots, \xi_{2,\tau_2-1}\}$. For $\imath \in \{0,1\}$, if $F_{1,\tau_1+\imath} = F_{1,\ell}$ for some $\ell \in \{0, \cdots, \tau_1 + \imath - 1\}$, then $\mathcal{B}$ sets $\xi_{1,\tau_1+\imath} = \xi_{1,\ell}$; otherwise, it sets $\xi_{1,\tau_1+\imath}$ to a random string in $\{0,1\}^* \setminus \{\xi_{1,0}, \cdots, \xi_{1,\tau_1+\imath-1}\}$. $\mathcal{B}$ sends $(\xi_{1,\tau_1}, \xi_{1,\tau_1+1}, \xi_{2,\tau_2})$ to $\mathcal{A}$, adding $(F_{1,\tau_1}, \xi_{1,\tau_1})$, $(F_{1,\tau_1+1}, \xi_{1,\tau_1+})$ to L_1, $(F_{2,\tau_2}, \xi_{2,\tau_2})$ to L_2. Finally, $\mathcal{B}$ adds 2 to τ_1, 1 to τ_2 and 1 to τ_s.

Let $\mathcal{A}$ query to $\mathcal{O}_X(\cdot)$ totally τ_s times with $\xi_{1,m_1}, \cdots, \xi_{1,m_{\tau_s}}$. The polynomials corresponding to elements in $\mathbb{G}_1$ that the adversary can compute as a linear combination of elements in its view are

$$F_{1,\alpha} = \alpha_0 + \sum_{i=1}^{\tau_s} \alpha_{1,i}(x + m_i s_i) + \sum_{i=1}^{\tau_s} \alpha_{2,i} \frac{1}{s_i + y} \tag{1}$$

where the coefficients $\alpha \in \mathbb{Z}_p$ are controlled by $\mathcal{A}$ and introduced after queries to $\mathcal{O}_X(\cdot)$ and the group operations in $\mathbb{G}_1$.

The polynomials corresponding to elements in $\mathbb{G}_2$ that the adversary can compute as a linear combination of elements in its view are

$$F_{2,\beta} = \beta_0 + \sum_{i=1}^{\tau_s} \beta_i s_i \tag{2}$$

here $\beta_i \in \mathbb{Z}_p$ are controlled by $\mathcal{A}$ and introduced after queries to $\mathcal{O}_X(\cdot)$ and the group operations in $\mathbb{G}_2$.

We drop the corresponding subscript i for clarity and have the following equations.

$$(s + y)F_{1,\alpha} = (\alpha_0 y + \alpha_2) + (\alpha_0 + \alpha_1 my)s + \alpha_1 xy + \alpha_1 ms^2 + \alpha_1 sx, \tag{1'}$$

$$(s + y)F_{2,\beta} = \beta_0 y + (\beta_1 y + \beta_0)s + \beta_1 s^2 \tag{2'}$$

Eventually $\mathcal{A}$ outputs a tuple of elements $(\xi_{1,m'}, \xi_{1,a}, \xi_{1,b}, \xi_{2,c})$, where $0 \leq m', a, b < \tau_1$ and $0 \leq c < \tau_2$. To later test the correctness of $\mathcal{A}$'s output within the framework of this game, $\mathcal{B}$ computes the polynomial:

$$F_{3,\Delta} = F_{1,a} - (x + m' F_{2,c}) \tag{3}$$

$$F_{3,\nabla} = F_{1,b}(F_{2,c} + y) - 1 \tag{4}$$

For $\mathcal{A}$'s response to always be correct, then $F_{3,\Delta} = F_{3,\nabla} \equiv 0$ holds for any value of $x \in \mathbb{Z}_p^*$. We argue that it is impossible for $\mathcal{A}$ to achieve this relation.

Now we have the following equations:

$$\begin{aligned}(s+y)F_{3,\Delta} &= (s+y)(F_{1,a} - (x + m'F_{2,c})) \\ &= a_2 + (a_0 + a_1 m - c_0 m')y + (a_0 - c_0 m')s + (a_1 m - c_1 m')ys \\ &\quad + (a_1 - 1)xy + (a_1 m - c_1 m')s^2 + (a_1 - 1)sx \end{aligned} \tag{5}$$

$$\begin{aligned}(s+y)F_{3,\nabla} &= (s+d)(F_{1,b}(F_{2,c} + y) - 1) \\ &= c_0 b_2 + (b_0 c_0 + b_2 c_1 - 1)s + (c_0 b_0 + b_2 - 1)y + (b_1 m c_0 + b_0 c_1)s^2 \\ &\quad + b_1 c_0 sx + (b_0 c_1 + b_1 c_0 m + b_0)sy + b_1 c_0 xy + b_1 c_1 m s^3 + \\ &\quad b_1 c_1 s^2 x + b_1(1 + c_1)m s^2 y + b_1(1 + c_1)sxy + b_0 y^2 + b_1 m s y^2 + \\ &\quad b_1 x y^2 \end{aligned} \tag{6}$$

To make $F_{3,\triangle} = F_{3,\nabla} \equiv 0$, the adversary cancels all the terms in both equations. In equation (6), to cancel terms y^2, xy^2, the adversary must set $b_0 = b_1 = 0$. To cancel term y, noting that $b_0 = 0$, the adversary must set $b_2 = 1$. As a result, to cancel term s, noting that $b_0 = 0, b_2 = 1$, the adversary must set $c_1 = 1$. Accordingly, to cancel the constant term $c_0 b_2$, the adversary must set $c_0 = 0$. In equation (5), to cancel term sx, the adversary must set $a_1 = 1$. However, to cancel term s^2, the adversary must set $m' = m$ which implies that $M' = \xi_{1,m'}$ is one of the queried message $M = \xi_{1,m}$.

One may note that, in the above, we have assumed that m is a constant. Since the adversary does not show m but ξ_m to the oracle, m can be adaptively related to $x, s_1, \frac{1}{s_1+y}, \cdots, s_{i-1}, \frac{1}{s_{i-1}+y}$, for the i-th query to $\mathcal{O}_X$. Indeed, it can be a linear combination of the above variables. Similarly, for simplicity, we drop the corresponding subscript and obtain that

$$m = m_0 + m_1 x + m_2 s + m_3 \tfrac{1}{s+y} \tag{$*$}$$

We show that this does influence the above proof. In equation 6, substituting m with $(*)$ does not introduce additional terms y^2, xy^2, y, s. So we still have $b_0 = b_1 = 0, b_2 = c_1 = 1, c_0 = 0$. In equation 5, noting that $c_0 = 0$, substituting m with $(*)$ does not introduce additional terms sx, s^2. Hence, the proof above treating m as a constant still holds. Therefore, $\mathcal{A}$ cannot make $F_{3,\triangle} = F_{3,\nabla} \equiv 0$ hold for any value of $(x, y, s_1, \cdots, s_{\tau_s})$. Thus, we conclude that $\mathcal{A}$'s success depends solely on its luck when $(x, y, s_1, \cdots, s_{\tau_s})$ is instantiated.

Now we analyze $\mathcal{B}$'s simulation. At this point $\mathcal{B}$ chooses random $z^* = (x^*, y^*, s_1^*, \cdots, s_{\tau_s}^*) \in (\mathbb{Z}_p^*)^{\tau_s+2}$. $\mathcal{B}$ now tests (in equations 7,8,9) if its simulation was perfect; that is, if the instantiation of z by z^* does not create any equality relation among the polynomials that were not revealed by the random strings provided to $\mathcal{A}$. $\mathcal{B}$ also tests (in equations 10 and 11) whether or not $\mathcal{A}$'s output was correct. Thus, $\mathcal{A}$'s overall success is bounded by the probability that any of the following holds:

$$F_{1,i}(z^*) - F_{1,j}(z^*) = 0, \text{ for some } i, j \text{ such that } F_{1,i} \neq F_{1,j}, \tag{7}$$

$$F_{2,i}(z^*) - F_{2,j}(z^*) = 0, \text{ for some } i, j \text{ such that } F_{2,i} \neq F_{2,j}, \tag{8}$$

$$F_{3,i}(z^*) - F_{3,j}(z^*) = 0, \text{ for some } i, j \text{ such that } F_{3,i} \neq F_{3,j}, \tag{9}$$

$$F_{3,\Delta}(z^*) = 0 \tag{10}$$

$$F_{3,\nabla}(z^*) = 0 \tag{11}$$

From equation $(*)$, the polynomials $F_{1,i}$, $F_{2,i}$, $F_{3,i}$, $F_{3,\Delta}$ and $F_{3,\nabla}(z^*)$ have respective degrees at most $\tau_s+2\tau+1$. For fixed i and j, the five cases occur with probability at most $(\tau_s + 2\tau + 1)/p$. Now summing over all (i, j) pairs in each case, we bound $\mathcal{A}$'s overall success probability $\varepsilon \leq \binom{\tau_1}{2}\frac{\tau_s+2\tau+1}{p} + \binom{\tau_2}{2}\frac{\tau_s+2\tau+1}{p} + \binom{\tau_3}{2}\frac{\tau_s+2\tau+1}{p} + \frac{(\tau_s+2\tau+1)^2}{p^2}$. Since $\tau_1 + \tau_2 + \tau_3 \leq \tau + 6$ and $\tau_s \leq \tau_2$, the required bound follows: $\varepsilon \leq 9(\tau + 6)^2\tau/p = O(\tau^3/p)$.

5.3 Extended Special Signature

We extend our special signature so that it can sign on an element in $\mathbb{G}_1$, together with block of messages in a commitment. We also derive efficient protocol for signature generations and zero-knowledge proof-of-knowledge of possession of a signature without revealing the message. We highlight the differences.

KeyGen. Pick additional generators $h_0, h_1, \cdots, h_L, u_0, u_1, u_2, u_4 \in \mathbb{G}_1$ and $u_3 \in \mathbb{G}_2$.

Sign. For any message M in $\mathbb{G}_1$ and block of messages $m_1, \cdots, m_L$ in $\mathbb{Z}_p^L$, randomly choose $r, s \leftarrow \mathbb{Z}_p^*$ and compute $a_1 = XM^r$, $a_2 = (g_1 h_0^s h_1^{m_1} \cdots h_L^{m_L})^{\frac{1}{y+r}}$, $a_3 = g_2^r$. The resulting signature is (a_1, a_2, a_3, s).

Signing Protocol. Suppose the block of messages to be signed is $(M, m_1, \cdots, m_L) \in (\mathbb{G}_1 \times \mathbb{Z}_p^L)$, the user compute the commitment as $C_m = h_0^{s'} h_1^{m_1} \cdots h_L^{m_L}$. The user sends M, C_m, together with a proof of knowledge of representation of C_m to the signer. The signer randomly picks s'' and computes $a_1 = XM^r, a_2 = (g_1 h_0^{s''} C_m)^{\frac{1}{y+r}}, a_3 = g_2^r$ and sends (a_1, a_2, a_3, s'') back to the user. The user computes $s = s' + s''$ and the signature on $M, m_1 \cdots, m_L$ is (a_1, a_2, a_3, s). In the protocol, the signer learns nothing about the block of messages $m_1, \cdots, m_L$ committed in C_m (but the signer do know M).

Verify. On input message M, a block of messages $(m_1, \cdots, m_L)$ and signature (a_1, a_2, a_3, r), output 1 if $e(a_1, g_2) = Ae(M, a_3)$ and $e(a_2, a_3 Y) = e(g_1, g_2)e(h_0, g_2)^s e(h_1, g_2)^{m_1} \cdots e(h_L, g_2)^{m_L}$. Output 0 otherwise.

Proof of Knowledge of Possession of a Signature Without Revealing the Message. On input message $M, (m_1, \cdots, m_L)$ and signature (a_1, a_2, a_3, s), compute the following quantities. $A_1 = a_1 u_1^{r_1}$, $A_2 = a_2 u_2^{r_2}$, $A_3 = a_3 u_3^{r_3}$, $A_4 = M u_4^{r_4}$, $A_5 = u_0^{r_3} u_1^{r_5}$ for some randomly generated $r_1, r_2, r_3, r_4, r_5 \in_R \mathbb{Z}_p^*$.

Compute the following proof of knowledge:

$$PK : \Big\{ (r_1, r_2, r_3, r_4, r_5, \delta_{2,3}, \delta_{2,5}, \delta_{3,4}, \delta_{4,5}, s, m_1, \cdots, m_L) : \\ A_5 = u_0^{r_3} u_1^{r_5} \ \wedge \ A_5^{r_2} = u_0^{\delta_{2,3}} u_1^{\delta_{2,5}} \ \wedge \ A_5^{r_4} = u_0^{\delta_{3,4}} u_1^{\delta_{4,5}} \ \wedge \\ \frac{Ae(A_4, A_3)}{e(A_1, g_2)} = \frac{e(A_4, u_3)^{r_3} e(u_4, A_3)^{r_4}}{e(u_4, u_3)^{\delta_{3,4}} e(u_1, g_2)^{r_1}} \ \wedge \\ \frac{e(A_2, A_3 Y)}{e(g_1, g_2)} = \frac{e(h_0, g_2)^s e(h_1, g_2)^{m_1} \cdots e(h_1, g_2)^{m_L} e(A_2, u_3)^{r_3} e(u_2, A_3 Y)^{r_2}}{e(u_2, u_3)^{\delta_{2,3}}} \Big\}$$

where $\delta_{2,3} = r_2 r_3, \delta_{2,5} = r_2 r_5, \delta_{3,4} = r_3 r_4, \delta_{4,5} = r_4 r_5$.

The verifier accepts the proof if the PK is valid.

It is straightforward to show that the above protocol is honest verifier zero-knowledge with special soundness. The extended special signature can also be proven to be existential unforgeable against adaptively chosen message attack under in the generic group model in a similar manner as the special signature. Unfortunately, we cannot reduce the security of these signature schemes to any existing hard problems. It remains an open problem to construction signature scheme with such properties under more standard assumptions in the standard model.

5.4 Bounded Accumulator

We make the observation that the accumulator due to [13] is in fact a bounded accumulator. We recall the details of the construction here.

BA-Setup. Assume $\mathbb{G}_1, \mathbb{G}_2$ is a bilinear group pair as discussed. Let g_1, g_2, q be generators of $\mathbb{G}_1$. Let $\alpha \in \mathbb{Z}_p^*$ be a secret seed. Let k be the upper bound of the bounded accumulator. Denote $q_i = q^{\alpha^i}$ for $i = 1, \cdots, k$. The public parameter of the bounded accumulator is then $(q, q_1, \cdots, q_k)$. The secret seed α can be deleted afterwards. In practice, BA-Setup should be run by a trusted party.

Accumulation. To accumulate a set of values $\{s_i\}$ for $i = 1, \cdots, k$ such that $s_i \in \mathbb{Z}_p^*$, compute $v = q^{\prod_{j=1}^{k}(\alpha - s_i)}$. Witness w_i of s_i such that s_i is a value accumulated in v is computed by $w_i = q^{\prod_{j=1, j\neq i}^{k}(\alpha - s_i)}$. Note that computation of v and w_i does not require the knowledge of α, by the use of $\{q_i\}$. Note that witness w_i for s_i satisfies $w_i^{\alpha + s_i} = v$.

Proof of Knowledge of a Value in an Accumulator. Let w_i be a witness of s_i for the accumulator v. To prove that s_i is in the accumulator, the prover computes the following quantities, $A_1 = g_1^{r_1} g_2^{r_2}$, $A_2 = w_i g_2^{r_1}$. The following zero-knowledge proof-of-knowledge protocol is then carried out.

$$PK : \Big\{ (r_1, r_2, \delta_1, \delta_2, s_i) : \\ A_1 = g_1^{r_1} g_2^{r_2} \ \wedge \ A_1^{s_i} = g_1^{\delta_1} g_2^{\delta_2} \ \wedge \ \frac{\hat{e}(v, q)}{\hat{e}(A_2, q_1)} = \hat{e}(A_2, q)^{s_i} \hat{e}(g_2, q)^{-\delta_1} \hat{e}(g_2, q_1)^{-r_1} \Big\}$$

where $\delta_1 = r_1 s_i, \delta_2 = r_2 s_i$.

Proof Without Revealing the Accumulator. The generic construction also requires the proof of knowledge of a value in an accumulator without revealing the accumulator. Here we propose the following protocol for such purpose. Again, let w_i be a witness of s_i for the accumulator v. To prove that s_i is in the accumulator, the prover computes the following quantities, $A_1 = g_1^{r_1} g_2^{r_2}$, $A_2 = w_i g_2^{r_1}$, $A_3 = v g_1^{r_3}$. The following zero-knowledge proof-of-knowledge protocol is then carried out.

$$PK : \Big\{ (r_1, r_2, r_3, \delta_1, \delta_2, s_i) : \\ A_1 = g_1^{r_1} g_2^{r_2} \ \wedge \ A_1^{s_i} = g_1^{\delta_1} g_2^{\delta_2} \ \wedge \\ \frac{\hat{e}(A_3, q)}{\hat{e}(A_2, q_1)} = \hat{e}(A_2, q)^{s_i} \hat{e}(g_2, q)^{-\delta_1} \hat{e}(g_2, q_1)^{-r_1} \hat{e}(g_1, q)^{r_3} \Big\}$$

where $\delta_1 = r_1 s_i, \delta_2 = r_2 s_i$.

Condition ACC: Condition ACC may not hold for this particular accumulator. In particular, if we set $g = q$, then given q^{s_1}, q^{s_2}, it is easy to tell if $v = q^{(\alpha+s_1)(\alpha+s_2)}$ given the knowledge of α. One solution is to make the XDH assumption in $\mathbb{G}_1$, the other assumption is to set $< g >$ to be a completely independent group where DDH is hard such that there is no efficiently computable isomorphism from $< g >$ to $\mathbb{G}_1$. The latter approach is employed to build our instantiation.

6 An Instantiation

Putting the building blocks together following the generic construction, we have an instantiation with full tracing. For completeness, the detail of the instantiation is shown in Appendix A.

We have the following theorem regarding the security of this particular instantiation. The proof is omitted due to the page limitation. We refer the reader to the full version of this paper for the detail.

Theorem 1. *Our compact e-cash with full tracing is secure under the q-SDH assumption and the SXDH assumption in the random oracle model and the generic group model.*

7 Concluding Remarks

Compared with CHL, our instantiation is more efficient in both spending and coin tracing. CHL has a noted advantage in the wallet size while our scheme has an edge in terms of bank's storage. Our instantiation requires an expensive accumulation process, which could be done offline and could be pre-computed. One of the problem of CHL is that it does not support concurrent withdrawal since the underlying CL/CL+ signature used does not support concurrent signature generation. We could not make the claim that our scheme supports concurrent withdrawal since our new signature scheme is secure in the generic group

model only. Having said that we proposed generic construction of compact e-cash scheme with private key revocation and full coin tracing from bounded accumulator, we also proposed an efficient instantiation. The special signature scheme we proposed may be of independent interest. Nonetheless, it remains an open problem to propose such signature scheme with more standard assumptions.

References

1. Giuseppe Ateniese, Jan Camenisch, and Breno de Medeiros. Untraceable rfid tags via insubvertible encryption. In *ACM Conference on Computer and Communications Security*, pages 92–101, 2005.
2. Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS*, 2005.
3. Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *CRYPTO*, pages 302–318, 1993.
4. Ernie Brickell, Peter Gemmell, and David Kravitz. Trustee-based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change. In *SODA '95: Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 457–466. Society for Industrial and Applied Mathematics, 1995.
5. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact E-Cash. In *EUROCRYPT*, pages 302–321, 2005.
6. Jan Camenisch and Anna Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *CRYPTO*, pages 61–76, 2002.
7. Sébastien Canard and Jacques Traoré. On fair e-cash systems based on group signature schemes. In *ACISP*, pages 237–248, 2003.
8. David Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*, pages 199–203. Plenum, New York, 1983.
9. Yevgeniy Dodis, Aggelos Kiayias, Antonio Nicolosi, and Victor Shoup. Anonymous identification in ad hoc groups. In *EUROCRYPT*, pages 609–626, 2004.
10. Oded Goldreich. Zero-Knowledge twenty years after its invention. Cryptology ePrint Archive, Report 2002/186, 2002. `http://eprint.iacr.org/`.
11. Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In *EUROCRYPT*, pages 571–589, 2004.
12. Greg Maitland and Colin Boyd. Fair Electronic Cash Based on a Group Signature Scheme. In *Information and Communications Security, Third International Conference, ICICS 2001, Xian, China, November 13-16, 2001*, volume 2229 of *Lecture Notes in Computer Science*, pages 461–465. Springer, 2001.
13. Lan Nguyen. Accumulators from Bilinear Pairings and Applications. In *CT-RSA*, pages 275–292, 2005.

A Instantiation

BankSetup. On input 1^λ, where λ is a security parameter, output $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ of order p where p is a λ-bit prime, with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$. Output generators g, g_1, g_2, g_3, q of $\mathbb{G}_1$ and $h, h_q, h_0, h_1, h_2, h_3$ of $\mathbb{G}_2$. Also output a group $\mathbb{G}_T$ of order p with u, u_1, u_2, u_3 as generator. Assume DDH is hard in $\mathbb{G}_T$. In order

to ensure that the generators are generated fairly, they maybe set to the output of some hash function on the bank's identity. The bank chooses $x, y, \alpha \in_R \mathbb{Z}_p^*$, computes $X = g^x, Y = h^y, Z = e(X, h), w = h_q^\alpha, q_i = q^{\alpha^i}$ for $i = 1, \cdots, k$. Also chooses a cryptographic hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_T$. The bank's public key is ($\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$, $\mathbb{G}_T$, e, H, u, u_1, g, h, q, g_1,g_2,g_3,h_q, h_0,h_1,h_2,h_3,w,$q_1, \cdots, q_k$, Y, Z) and the secret key is (X, y). α can be safely deleted.
Remarks: It is straightforward to show that, with the knowledge of α, anyone can over-spends their wallets without being detected. On the other hand, α cannot help breaking the anonymity nor can it helps to slander an honest user. Keeping α is, thus, exactly against the interest of the bank and we can assume that the bank will delete this information honestly.

UserSetup. Each user is equipped with discrete logarithm type key pairs $(d, u^d) \in \mathbb{Z}_p^* \times \mathbb{G}_T$.

Withdrawal Protocol. User with public key PK computes a set of k random values $\{s_i\}$ and $V = q^{\prod_{j=1}^{k}(\alpha - s_i)}$. He also computes the set of k witnesses $\{w_i\}$. He then computes $C_d = g_0^{r'} g_1^d g_2^{tr'}$ and sends V, C_d to the bank. The user is required to prove to the bank that C_d is correctly formed by executing $PK\{(d, tr') : C_d = g_0^{tr'} g_1^d \ \wedge \ \mathrm{PK} = u^d\}$. The bank randomly picks tr'', r, computes $a_1 = XV^r$ and $a_2 = (g g_0^{tr''} C_d)^{\frac{1}{y+r}}$, $a_3 = h^r$. The bank sends (tr'', a_1, a_2, a_3) back to the user. The user computes $tr = tr' + tr''$. The user wallet is $(a_1, a_2, a_3, tr, d, \{s_i\}, \{w_i\})$. The user also needs to verifiable encrypts tr under its public key as follows. Compute $T_0 = u^{r_1} u_1^{tr}, T_1 = u^{r_2} u_1^d$. Prove that T_0, T_1 are correctly formed with respect to $g_0^{tr''} C_d$ and compute $z_r = r_1 - c r_2, z_{tr} = tr - cd$ with $c = H(T_0, T_1, g_0^{tr''} C_d)$. (z_{tr}, c) is the verifiable encryption on tr and is saved in the banks database.

Spend Protocol. The user and the merchant with identity $I \in \{0,1\}^*$ first agree on the transaction information info. The user then chooses one pair of the unused s_i, w_i in his wallet and computes the following quantities, $S = u_1^{s_i}$, $T = u_1^{k_1} u_2^{s_i}$, $U = u_1^{k_2} u_2^d$, $Tag = H(I||\mathrm{info})^{tr}$ for some randomly generated $k_1, k_2 \in_R \mathbb{Z}_p^*$. The user also computes $A_1 = a_1 g_1^{r_1}$, $A_2 = a_2 g_2^{r_2}$, $A_3 = a_3 h_0^{r_3}$, $A_4 = V g_3^{r_4}$,$A_5 = g_0^{r_3} g_1^{r_5}$, $B_1 = g_0^{r_6} g_1^{r_7}$, $B_2 = w_i g_2^{r_7}$ for some randomly generated $r_1, r_2, r_3, r_4, r_5, r_6, r_7 \in_R \mathbb{Z}_p^*$. Compute the following signature of knowledge:

$$\pi_{spend} : SPK\Big\{(k_1, k_2, r_1, r_2, r_3, r_4, r_5, r_6, r_7, \delta_{2,3}, \delta_{2,5}, \delta_{3,4}, \delta_{3,5}, \delta_6, \delta_7, s_i, d, tr) :$$
$$A_5 = g_0^{r_3} g_1^{r_5} \ \wedge \ A_5^{r_2} = g_0^{\delta_{2,3}} g_1^{\delta_{2,5}} \ \wedge \ A_5^{r_4} = g_0^{\delta_{3,4}} g_1^{\delta_{4,5}} \ \wedge \ B_1 = g_0^{r_6} g_1^{r_7} \ \wedge$$
$$B_1^{s_i} = g_0^{\delta_6} g_1^{\delta_7} \ \wedge \ \frac{e(A_4, A_3)}{e(A_1, h)} = \frac{e(A_4, h_0)^{r_3} e(g_3, A_3)^{r_4}}{e(g_1, h)^{r_1} e(g_3, A_3)^{r_4}} \ \wedge$$
$$\frac{e(A_2, A_3 Y)}{e(g, h)} = \frac{e(g_0, h)^{tr} e(g_1, h)^d e(g_2, A_3)^{r_2} e(A_2, h_0)^{r_3}}{e(g_2, h_0)^{\delta_{2,3}}} \ \wedge$$
$$\frac{e(A_4, h_q)}{e(B_2, w)} = \frac{e(B_2, h_q)^{s_i} e(g_3, h_q)^{r_4}}{e(g_2, w)^{r_7} e(g_2, h_q)^{\delta_7}} \ \wedge$$
$$S = u_1^{s_i} \ \wedge \ T = u_1^{k_1} u_2^{s_i} \ \wedge \ U = u_1^{k_2} u_2^d \ \wedge \ Tag = H(I||\mathrm{Info})^{tr}\Big\}(I||\mathrm{Info})$$

where $\delta_{2,4} = r_2 r_4, \delta_{2,5} = r_2 r_5, \delta_{3,4} = r_3 r_4, \delta_{4,5} = r_4 r_5, \delta_6 = r_6 s_i, \delta_7 = r_7 s_i$

π_{spend} can be abstracted as the following signature of knowledge.

$$SPK\Big\{(a_1, a_2, a_3, w_i, s_i, V, k_1, k_2, d, tr) :$$

$$e(a_1, h) = Ze(V, a_3) \wedge \ e(a_3, a_3 Y) = e(g g_0^{tr} g_1^d, h) \wedge \ w_i^{\alpha + s_i} = V \wedge \ S = u_1^{s_i} \wedge$$
$$T = u_1^{k_1} u_2^{s_i} \wedge \ U = u_1^{k_2} u_2^d \wedge \ Tag = H(I||\text{info})^{tr} \Big\}(I||\text{Info})$$

The user also computes $(z_k = k_1 - ck_2, z_d = s_i - cd)$, where c is the challenge used in π_{spend}. The user then sends (π_{spend}, z_k, z_d) to the merchant. The merchant accept the payment if π_{spend} is valid and $T = U^c g^{z_k} h^{z_d}$. The merchant also After the payment is accepted, the user can delete s_i, w_i.

Deposit and RevokeDoubleSpender have been described in the generic construction.

Full Tracing. After computing d such that u^d is the public key of the double-spender, the bank computes all tr's during the withdrawal protocols of the same user. This is done by $tr = z_{tr} + cx$ where (z_{tr}, c) is retrieved from the database. The bank publishes tr for all shops to check the spendings. To trace past spendings, the banks needs to run through its database and see if $Tag = H(I||\text{info})^{tr}$.

Batch Processing of Interactive Proofs

Koji Chida and Go Yamamoto

NTT Information Sharing Platform Laboratories, NTT Corporation,
1-1 Hikarinooka, Yokosuka-shi 239-0847 Japan

Abstract. We present a new design principle for building a batch processing protocol for interactive proofs. First, a generic method to achieve batch processing is proposed when dealing with an NP-relation with certain homomorphicity. It is shown that the method preserves zero-knowledgeness and knowledge-soundness. Second, for some NP-relation that has no such homomorphicity, we illustrate that the relation can be decomposed into a homomorphic relation(hence we have a batch process) and another NP-relation that is proven using an efficient protocol. Such a decomposition provides an advantage in terms of efficiency.

Keywords: Batch Processing, Proofs of Knowledge, Sigma-Protocols.

1 Introduction

Interactive proofs are pairs of algorithms with interactions between a prover and a verifier. Usually the provers and the verifiers are modeled as a probabilistic polynomial-time Turing machine. Interactive proofs with zero-knowledgeness and knowledge-soundness are useful components for various identification schemes and multi-party protocols.

In actual applications, it is often the case that a single algorithm is reiterated many times. For example, many digital signatures are sometimes verified by a single trusted verifier. Thus, it is important to improve the performance of interactive proofs when a protocol is reiterated many times.

The issue addressed herein is how to reduce the amount of resources required by the provers and verifiers when a single interactive proof is reiterated for n distinct instances. Obviously, if the algorithm is simply reiterated n times, then the required resources are also as much as n times. However, a better efficiency can be achieved for many cases. For example [10] proposes an algorithm that verifies many instances of protocol efficiency. Reference [3] describes how to reduce multiplications in exponentiations using precomputation. Reference [1] suggests a more efficient algorithm for a verifier with modular exponents. Reference [5] proposes a batch processing protocol for the Schnorr identification scheme, which reduces not only the computational complexity, but also the amount of communications traffic. It is a unique and important feature that batch processing can reduce both the computational complexity and the amount of communications. The following subsections examine this more closely.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 196–207, 2007.

Batch Processing for the Schnorr Protocol. Let g be an element of finite abelian group G, whose order is a sufficiently large prime $p = |g|$. The protocol described below is well-known [12].

Protocol 1
Private input to *P:* $x \in \mathbb{Z}/p\mathbb{Z}$,
Common input: $y = g^x$

1. *P chooses* $r \leftarrow \{0, .., p-1\}$ *randomly, sends* $R = g^r$ *to V.*
2. *V chooses* $c \leftarrow \{0, .., p-1\}$ *randomly, and sends it to P.*
3. *P computes* $z = r - xc \mod p$, *and sends it to V.*
4. *V checks* $g^z = y^{-c}R$. *If successful, accept the proof.*

This protocol is honest verifier zero-knowledge and is knowledge-sound, as defined in [8] for example.

Protocol 1 is often used as a part of identification schemes, signature schemes, and multi-party protocols. In such situations, the protocol may be reiterated many times.

Let us suppose that there are d provers and a verifier. In this case, the computational resources of the verifier may represent a bottleneck in the system. Reference [6] proposes a technology to improve the computational efficiency compared to a scheme where the verifier simply repeats the verification step d times. This approach is called batch verification.

There are some ways to improve the performance if there is a single prover and a single verifier, where the application requests processing of Protocol 1 for d instances. Reference [5] shows that not only can the verifier computational complexity be reduced, but also the prover computational complexity and the amount of communications traffic. Here we describe the protocol.

Protocol 2
Private input to *P:* $x_i \in \mathbb{Z}/p\mathbb{Z} (i = 1, 2, \cdots, d)$,
Common input: $y_i = g^{x_i}$

1. *P chooses* $r \leftarrow \{0, .., p-1\}$ *randomly, sends* $R = g^r$ *to V.*
2. *V chooses* $c_i \leftarrow \{0, .., p-1\}$ *randomly for each* $i = 1, 2, \cdots, d$, *then sends them to P.*
3. *P computes* $z = w - \sum_i x_i c_i \mod p$, *and sends it to V.*
4. *V checks* $g^z = R \prod_i y_i^{-c_i}$. *If successful, accept the proof.*

Protocol 2 is honest verifier zero-knowledge and is a proof of knowledge for $(x_1, x_2, \cdots, x_d)$. Reference [7] suggests that V may send only $c \leftarrow \{0, .., p-1\}$ and set $c_i = c^i \mod p$. Since this approach processes many instances of the entire protocol at once, we call it *batch processing*. In the above protocol, the number of prover exponentiations is reduced to a constant based on d. The

amount of communications traffic is also reduced to a constant based on d, since c_i may be generated from a single c.

We observe Protocol 2 to obtain the essential point for batch processing: "Mix" the instances using the verifier public coin, so as not to "degenerate" them.

Our Contribution. This paper presents a design principle to construct batch processing protocols for NP-relations with certain homomorphicity, which we call *additivity*. We will show that if interactive proof π is knowledge-sound, then the batch processing protocol is also knowledge-sound.

Nonetheless, batch processing can be applied when dealing with some non-additive NP-relation. Our second idea is that a non-additive NP-relation can be "decomposed" into two statements: One is additive, the other is not necessarily additive but easy to prove. We illustrate the idea for the proof in a multi-party protocol in [13], which yields a new protocol for batch processing with advantages in terms of both computational complexity and communications traffic over simple repetition of the known protocol.

2 Additive NP-Relations and Batch Processing

Let us consider NP-relation R and protocol π between prover P and verifier V where π is an interactive proof with knowledge-soundness.

Definition 1. *Let R be an NP-relation and let $L_R = \{y \in \{0,1\}^* | (y,x) \in R$ for some $x \in \{0,1\}^*\}$, $W(y) = \{x \in \{0,1\}^* |\ (y,x) \in R\}$, and $W = \cup_y W(y)$. Suppose L_R is an abelian group with polynomial-time algorithms for the operations and suppose that on input x_1, x_2 where $x_1 \in W(y_1), x_2 \in W(y_2)$ a polynomial-time algorithm outputs some x such that $x \in W(y_1 y_2)$. Then we call R* additive. *Moreover, if L_R is a non-trivial $\mathbb{Z}/p\mathbb{Z}$-module, then we call R p-*additive *where p is a prime number.*

Example 1. – Let $R = \{(y,x) |\ y = g^x\}$. Then R is p-additive.
– Let $R = \{((y_0, y_1), x) |\ y_0 = g^x$ and $y_1 = h^x\}$. Then R is p-additive.

Note 1. For additive NP-relation R, if there exists a randomized algorithm that outputs $(y,x) \in R$ such that y is uniform in L_R, then R is random self-reducible [14].

Note 2. Even though the set of instances has an associative multiplication, the set of witnesses may not necessarily have an associative multiplication. Let $(y_1, x_1), (y_2, x_2), (y_3, x_3) \in R$ and compute $x_{12} \in W(y_1 y_2)$ and then $x_{(12)3} \in W((y_1 y_2) y_3)$. In the same way compute $x_{1(23)}$. Then, $x_{(12)3}$ and $x_{1(23)}$ are not guaranteed to coincide.

For $(y,x) \in R$, we denote $\pi(y;x)$ the interactive proof between P and V, where y is the common input, and x is the private input to P.

Let $d = d(k)$ be a family of numbers bounded by k^γ where k is the security parameter for π and γ is a constant. Let R be p-additive. Then consider

the protocol $\pi^d(y_1, y_2, \cdots, y_d; x_1, x_2, \cdots, x_d)$ as described below where (y_1, x_1), $(y_2, x_2), \cdots, (y_d, x_d) \in R$. We will show that π^d processes d instances of π.

Protocol 3
Private input to *P:* $x_i \in \mathbb{Z}/p\mathbb{Z}(i = 1, 2, \cdots, d)$,
Common input: $y_i(i = 1, 2, \cdots, d)$

1. *V chooses* $e \leftarrow \{0, .., p-1\}$ *randomly, and sends it to P.*
2. *Run* $\pi(\prod_i y_i^{e^{i-1} \mod p}; x)$ *where* $x \in W(\prod_i y_i^{e^{i-1} \mod p})$. *If accepted, V accepts the proof.*

Then we have the following theorems.

Theorem 1. *If* π *is (perfect, statistical, computational) zero-knowledge, then* π^d *is (perfect, statistical, computational)zero-knowledge.*

The proof is obvious.

Theorem 2. *If* π *is knowledge-sound, then* π^d *is knowledge sound.*

Proof. Let P^* be a possibly cheating prover for π^d. Set probability P^* and make V accept the proof $p(y_1, y_2, .., y_d) = \text{Prob}[(P^*, V) \in \text{Acc}]$ where (P^*, V) is a transcript between P^* and V.

For each $e \in \mathbb{Z}/p\mathbb{Z}$, let V_e be the same algorithm that the verifier processes in π^d except for the first message where V_e always outputs e.

Let K_P be the knowledge extractor for π. The expected running time for K is bounded by $\frac{k^c}{p(y_1, y_2, .., y_d) - \varepsilon(k)}$ where ε is a negligible function, the knowledge error for π.

Consider the P^*-oracle machine as described below.

Algorithm 1

1. $i \leftarrow 1$,
2. *Choose* $e \leftarrow \{0, .., p-1\}$ *randomly. Run* π *between* P^* *and* V_e. *If not accepted, repeat Step 2.*
3. *If* $e \in \{e_1, e_2, \cdots, e_{i-1}\}$ *then go to Step 2.*
4. *Run* K_{P^*} *until the running time reaches* $\frac{4k^c}{p(y_1, y_2, .., y_d) - \varepsilon(k)}$. *If* K_{P^*} *outputs x such that* $(\prod_j y_j^{e^{j-1} \mod p}, x) \in R$, *then proceed to the next step. Otherwise go to Step 2.*
5. *Set* $e_i \leftarrow e$, $x_i \leftarrow x$, $\bar{y}_i = \prod_j y_j^{e^{j-1} \mod p}$, *and* $i \leftarrow i + 1$. *If* $i \leq d$, *then go to Step 2.*
6. *Let E be a* $d \times d$ *matrix value on* $\mathbf{F}_p$ *defined by* $E_{i,j} = e_i^{j-1}$. *If* $\det(E) = 0$, *then go to Step 1. Otherwise compute* $\hat{x}_i$ *such that* $(\prod_j \bar{y}_j^{(E^{-1})_{i,j}}, \hat{x}_i) \in R$ *for each* $i = 1, 2, \cdots, d$. *Output* $(\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_d)$.

Consider $\mathcal{E} = \{e |\ \Pr[(P^*, V_e) \in \text{Acc}] > \frac{p(y_1, y_2, .., y_d)}{2}\}$. A simple counting argument proves that $\Pr[e \leftarrow_R \{0, 1, \cdots, p-1\}, (P^*, V_e) \in \text{Acc} : e \in \mathcal{E}] > \frac{1}{2}$.

Then, Step 2 chooses e from $\mathcal{E}$ with the probability of at least $\frac{1}{2}$. If $e \in \mathcal{E}$, then K_{P^*} outputs x such that $g^x = \prod_j y_j^{e^{j-1} \mod p}$ in $\frac{4k^c}{p(y_1,y_2,..,y_d)-\varepsilon(k)}$ steps with the probability of at least $\frac{1}{2}$.

Matrix E is non-degenerative if and only if $e_1, e_2, \cdots, e_d$ are distinct in $\mathbf{F}_p$ to each other. It is clear that $\prod_j \bar{y}_j^{(E^{-1})_{i,j}} = y_i$. Thus, we obtain the output in $\frac{4d(4k^c+1)}{p(y_1,y_2,..,y_d)-\varepsilon'(k)}$ expected steps, where ε' is a negligible function. ■

3 Application to the Guillou-Quisquater Identification Scheme

We present a useful application for Protocol 3.

Reference [9] presents an identification scheme described in Protocol 4, where P proves its identity. Let ν be a prime number and $N = pq$ be an RSA modulus.

Protocol 4 *([9])*
Private key: $B \in (\mathbb{Z}/N\mathbb{Z})^*$
Public key:$J = B^{-\nu}$

1. *P chooses $r \in \{1, 2, \cdots, N-1\}$ randomly, computes $T = r^\nu$, and sends T to V.*
2. *V chooses $c \in \{0, 1, \cdots, \nu - 1\}$ randomly. V sends c to P.*
3. *P compute $D = rB^c$ and sends D to V.*
4. *V checks $T = D^\nu J^c$. If successful, then V accepts the proof.*

Fix $J_1, J_2, \cdots, J_d \in (\mathbb{Z}/N\mathbb{Z})^*$. Consider NP-relation $R \subset (\mathbb{Z}/\nu\mathbb{Z})^d \times (\mathbb{Z}/N\mathbb{Z})^*$ defined by $R = \{((t_1, t_2, \cdots, t_d), B) | \prod_i J_i^{t_i} = B^\nu\}$. It is clear that R is ν-additive and Protocol 4 proves the knowledge of R.

Apply Protocol 3 where we set π in Protocol 4. Then we obtain the protocol below that batch-processes Protocol 4, which is left open in [7].

Protocol 5
Private key: $B_i \in (\mathbb{Z}/N\mathbb{Z})^*$
Public key:$J_i = B_i^{-\nu}$

1. *P chooses $r \in \{1, 2, \cdots, N-1\}$ randomly. Then, P computes $T = r^\nu$ and sends T to V.*
2. *V chooses $e, c \in \{0, 1, \cdots, \nu - 1\}$ randomly. V sends eandc to P.*
3. *P compute $D = r(\prod_i B_i^{ce^{i-1} \mod \nu})$ and sends D to V.*
4. *V checks $T = D^\nu(\prod_i J_i^{ce^{i-1} \mod \nu})$. If successful, then V accepts the proof.*

Table 1. Performance Comparison for Batched Schnorr, GQ, and Batched GQ

	P multiplications	V multiplications	communications
Batched Schnorr	$\frac{3}{2}k$	$3k + kd$	$4k$
GQ	$\frac{3}{2}kd$	$\frac{5}{2}kd$	$(\|N\| + k)d$
Batched GQ	$k + \frac{1}{2}kd$	$2k + \frac{1}{2}kd$	$\|N\| + k$

We moved the step where V sends e to P to Step 2, since in this case, we are dealing with honest verifier ZK and the first message from P does not depend on e. In practice e can be determined by c.

Table 1 shows a performance comparison for the batched Schnorr scheme, GQ scheme, and batched GQ scheme where we count the anticipated number of multiplications in exponentiations using a binary method. We set $|p| = 2k$ and ν is a prime number close to 2^k where k is the security parameter. Although the batched Schnorr scheme is advantageous in regard to the prover efficiency and communications traffic, if d is sufficiently large, the batched GQ scheme is advantageous in regard to the verifier efficiency.

4 Application to an ID-Based Identification Scheme

Let E be an elliptic curve with Weil pairing $e_l : E[l] \times E[l] \to G$, where l is a prime number, G is an abelian group of order l, and $E[l]$ is the set of l-torsion points. Let $g, h \in E[l]$. The protocol below, described in [2], provides the ID-based identification scheme.

Protocol 6
Private key: $x \in E[l]$
Public key:$y \in E[l]$

1. *P chooses $k \in \{0, 1, \cdots, l-1\}$ randomly. Then, P computes $r = e_l(y, g)^k$ and sends r to V.*
2. *V chooses $c \in \{0, 1, \cdots, l-1\}$ randomly. V sends c to P.*
3. *P computes $s = cx + ky$ and sends s to V.*
4. *V checks $r = e_l(s, g)e_l(y, h)^{-c}$. If successful, then V accepts the proof.*

It is shown that Protocol 6 is a proof of knowledge with regard to relation

$$R = \{(y, x) \in E[l] \times E[l] |\ e_l(x, g) = e_l(y, h)\}.$$

Then, we observe that relation R is an l-additive NP-relation.

Proposition 1. *Relation R is l-additive.*

Table 2. Performance Comparison Between ID-Schnorr and Batched ID-Schnorr

	V pairings	communications
ID-Schnorr	$2d$	$(2\|l\|+t)d$
Batched ID-Schnorr	2	$3\|l\|+t$

Proof. Suppose $(y_1, x_1), (y_2, x_2) \in R$. Since $e_l(x_1, g) = e_l(y_1, h)$ and $e_l(x_2, g) = e_l(y_2, h)$, we have $e_l(x_1+x_2, g) = e_l(y_1+y_2, h)$, thus R is additive. R is l-additive since $R \subset E[l] \times E[l]$. ■

We apply Protocol 3 where we set π Protocol 6 to obtain Protocol 7 below, which batch-processes Protocol 6.

Protocol 7
Private key: $x_i \in E[l](i = 1, 2, \cdots, d)$
Public key:$y_i \in E[l](i = 1, 2, \cdots, d)$

1. *V chooses $e \in \{0, 1, \cdots, l-1\}$ randomly. Then V sends e to P.*
2. *P chooses $k \in \{0, 1, \cdots, l-1\}$ randomly. Then P computes $r = \prod_i e_l(y_i, g)^{ke^{i-1}}$ and sends r to V.*
3. *V chooses $c \in \{0, 1, \cdots, l-1\}$ randomly. V sends c to P.*
4. *P computes $s = \sum_i e^{i-1}(cx_i + ky_i)$ and sends s to V.*
5. *V checks $r = e_l(s, g)e_l(\sum_i e^{i-1}y_i, h)^{-c}$. If successful, then V accepts the proof.*

In this case, we cannot move the first message, e sent by V, to Step 3, since the first message sent by P depends on e. This implies that to convert Protocol 7 into a signature scheme replacing c with an output of a hash function, P must obtain nonce e from the outside every time he signs.

Table 2 shows a performance comparison between the ID-Schnorr scheme and the batched ID-Schnorr scheme where we assume that P will precompute $e_l(y_i, g)$. We set t to the length of a bit string that represents points on $E[l]$. We observe that the communications and the number of computations for pairings do not depend on d.

5 Application to a Multi-party Protocol for Secure Circuit Evaluation

We present another example that deals with non-additive NP-relation R where we apply Protocol 3 by decomposing R.

Reference [13] provides a multi-party protocol for secure circuit evaluation. In [13], a party is requested to prove his correctness by computing Pedersen commitment [11], essentially by the protocol below.

Let $g, h \in \mathcal{G}$ be common bases for $\mathcal{G}$, and $X_i, Y_i, G_i, H_i \in \mathcal{G}$ is generated randomly for each $i = 1, 2, \cdots, m$, where m is the number of parties. It is assumed that it is hard to compute u_i such that $g^{u_0} \prod_i X_i^{u_i} = 1$ and so on.

Protocol 8 *([13])*
Private input to P**:** $(s, t, w, x) \in (\mathbb{Z}/p\mathbb{Z})^4$
Common input:$i \in \{1, 2, \cdots, m\}$,
$(Z, \bar{G}, \bar{H}, \bar{X}, \bar{Y}) = (g^x h'^w, g^s G_i^x, h^s H_i^x, g^t X_i^x, h^t Y_i^x) \in \mathcal{G}^5$

1. *P chooses $\delta, \eta, \rho, \tau \in_R \mathbb{Z}/p\mathbb{Z}$ randomly, computes*
$$A = g^\eta h'^\rho,\ B = g^\delta G_i^\eta,\ C = h^\delta H_i^\eta,$$
$$D = g^\tau X_i^\eta,\ E = h^\tau Y_i^\eta,$$
and sends (A, B, C, D, E) to V.
2. *V chooses $c \in \mathbb{Z}/p\mathbb{Z}$ randomly.*
3. *P computes*
$$z_1 = \eta - cx \bmod p,\ z_2 = \rho - cw \bmod p,$$
$$z_3 = \delta - cs \bmod p,\ z_4 = \tau - ct \bmod p,$$
and sends (c, z_1, z_2, z_3, z_4) to V.
4. *V accepts the proof if*
$$A = g^{z_1} h'^{z_2} Z^c,\ B = g^{z_3} G_i^{z_1} \bar{G}^c,\ C = h^{z_3} H_i^{z_1} \bar{H}^c$$
$$D = g^{z_4} X_i^{z_1} \bar{X}^c,\ \text{and}\ E = h^{z_4} Y_i^{z_1} \bar{Y}^c.$$

Protocol 8 is a proof of knowledge for NP-relation

$$R = \{((i, Z, \bar{G}, \bar{H}, \bar{X}, \bar{Y}), (w, s, t, x) |$$

$$(Z, \bar{G}, \bar{H}, \bar{X}, \bar{Y}) = (g^x h'^w, g^s G_i^x, h^s H_i^x, g^t X_i^x, h^t Y_i^x)\}.$$

It is clear that R is not additive.

According to the observation in Section 3, we should "decompose" R into R_1 and R_2, where R_1 is additive, and R_2 is not additive but easy to prove.

Here we set R_1 and R_2 as given below.

$$R_1 = \{\big((Z, \bar{G}, \bar{H}, \bar{X}, \bar{Y}), (w, s, t, x_1, \cdots, x_m)\big) \mid (Z, \bar{G}, \bar{H}, \bar{X}, \bar{Y}) =$$
$$(g^{\sum_i x_i} h'^w, g^s \prod_i G_i^{x_i}, h^s \prod_i H_i^{x_i},\ g^t \prod_i X_i^{x_i}, h^t \prod_i Y_i^{x_i})\}.$$

$$R_2 = \left\{\big((i, \bar{G}), (\hat{s}, \hat{x})\big) \mid\ \bar{G} = g^{\hat{s}} G_i^{\hat{x}}\right\}.$$

It is clear that R_1 is p-additive. Thus, we design the batch processing of proofs of knowledge for R according to the principle shown below.

Protocol 9
Private input to P **:** $(s(j), t(j), w(j), x(j)) \in (\mathbb{Z}/p\mathbb{Z})^4$
Common input: $i(j) \in \{1, 2, \cdots, m\}$, $(Z(j), \bar{G}(j), \bar{H}(j), \bar{X}(j), \bar{Y}(j))$
$= (g^{x(j)} h'^{w(j)}, g^{s(j)} G_{i(j)}^{x(j)}, h^{s(j)} H_{i(j)}^{x(j)}, g^{t(j)} X_{i(j)}^{x(j)}, h^{t(j)} Y_{i(j)}^{x(j)}) \in \mathcal{G}^5$,
$j = 1, \ldots, d$

1. *Make proofs of knowledge on* R_2 *for* $(i(j), \bar{G}(j))$ *for each* $j = 1, 2, \cdots, d$.
2. *Make batch processing of proofs of knowledge on* R_1 *for* $(Z(j), \bar{G}(j), \bar{H}(j), \bar{X}(j), \bar{Y}(j))_j$, *where* $j = 1, 2, \cdots, d$.
3. *If both Step 1 and Step 2 are accepted, then accept the proof.*

Proposition 2. *Protocol 9 is knowledge-sound, if both proofs in Step 1 and in Step 2 are knowledge-sound.*

Proof. Suppose we have knowledge extractors K_1 and K_2 for Step 1 and for Step 2 respectively. Then for each $j = 1, 2, \cdots, m$ we obtain $(w(j), s(j), t(j), x_1(j), \cdots, x_m(j))$ such that $(Z(j), \bar{G}(j), \bar{H}(j), \bar{X}(j), \bar{Y}(j)) = (g^{\sum_i x_i(j)} h'^{w(j)}, g^{s(j)} \prod_i G_i^{x_i(j)}, h^{s(j)} \prod_i H_i^{x_i(j)}, \quad g^{t(j)} \prod_i X_i^{x_i(j)}, h^{t(j)} \prod_i Y_i^{x_i(j)})$ and $(\hat{s}(j), \hat{x}(j))$ such that $\bar{G}(j) = g^{\hat{s}(j)} G_{i(j)}^{\hat{x}(j)}$. Hence, we have $g^{s(j)-\hat{s}(j)} G_{i(j)}^{x_{i(j)}-\hat{x}(j)} \prod_{i \neq i(j)} G_i^{x_i(j)} = 1$. Since $g, G_1(j), G_2(j), \cdots, G_d(j)$ is chosen independently and at random, we have $s(j) = \hat{s}(j)$, $x_i(j) = \hat{x}(j)$ if $i = i(j)$, otherwise $x_i(j) = 0$ by non-negligible probability. Thus, our knowledge-extractor for Protocol 9 suffices to output $(s(j), t(j), w(j), x_{i(j)}(j))$ for each $j = 1, 2, \cdots, m$. ■

Here, we start to establish Step 1 and Step 2. Proofs of knowledge for R_2 are well-known. They are given for convenience.

Protocol 10
Private input to P**:** $(s, x) \in (\mathbb{Z}/p\mathbb{Z})^4$
Common input: $i \in \{1, 2, \cdots, m\}$, $\bar{G}{=}g^s G_i^x \in \mathcal{G}$

1. P *chooses* $\alpha, \beta \in_R \mathbb{Z}/p\mathbb{Z}$ *randomly, computes* $F = g^\alpha G^\beta$, *and sends* F *to* V.
2. V *chooses* $c \in \mathbb{Z}/p\mathbb{Z}$ *randomly.*
3. P *computes* $z_0 = \alpha - cs \bmod p$, $z_1 = \beta - cx \bmod p$, *and sends* (z_0, z_1) *to* V.
4. V *accepts the proof if* $F = g^{z_0} G^{z_1} \bar{G}^c$.

A protocol for R_1 is designed in a straight-forward way. Since R_1 is additive, we obtain a batch processing protocol as described in Protocol 11.

Table 3. Performance Comparison Between Simple Protocol and Batch Processing

	P exponentiations	V exponentiations	communications
Protocol 8	$10d$	$15d$	$5d\|p\|$
Protocol 12	$6d+6$	$12d+6$	$(3d+4)\|p\|$

Protocol 11
Private input to P: $(s(j), t(j), w(j), x_1(j), \cdots, x_m(j)) \in (\mathbb{Z}/p\mathbb{Z})^4$
Common input: $\mathcal{G}$, p, g, $(h, h') \in \mathcal{G}^2$, $(G_i, H_i, X_i, Y_i) \in \mathcal{G}^4$, $(Z(j), \bar{G}(j), \bar{H}(j), \bar{X}(j), \bar{Y}(j)) \in \mathcal{G}^5$, *where* $j = 1, 2, \cdots, d$.

1. *V chooses* $e \leftarrow \mathbb{Z}/p\mathbb{Z}$ *randomly.*
2. *P chooses* $\delta, \eta_i, \rho, \tau \in_R \mathbb{Z}/p\mathbb{Z}$ *for each* $i = 1, \ldots, m$ *randomly, computes*
$$A = g^{\sum_i \eta_i} h'^{\rho},\ B = g^{\delta} \prod_i G_i^{\eta_i},$$
$$C = h^{\delta} \prod_i H_i^{\eta_i},\ D = g^{\tau} \prod_i X_i^{\eta_i},$$
$$E = h^{\tau} \prod_i Y_i^{\eta_i},$$
and sends (A, B, C, D, E) *to* V.
3. *V chooses* $c \leftarrow \mathbb{Z}/p\mathbb{Z}$ *randomly.*
4. *P computes* $z_{1,i} = \eta_i - c \sum_j e^{j-1} x_i(j) \bmod p$,
$z_2 = \rho - c \sum_j e^{j-1} w(j) \bmod p$, $z_3 = \delta - c \sum_j e^{j-1} s(j) \bmod p$,
$z_4 = \tau - c \sum_j e^{j-1} t(j) \bmod p$, $\quad (i = 1, \ldots, d)$,
and then sends $(z_{1,1}, \ldots, z_{1,m}, z_2, z_3, z_4)$ *to* V.
5. *V accepts if*
$$A = g^{\sum_i z_{1,i}} h'^{z_2} \prod_j Z(j)^{ce^{j-1}}$$
$$B = g^{z_3} \prod_i G_i^{z_{1,i}} \prod_j \bar{G}(j)^{ce^{j-1}},$$
$$C = h^{z_3} \prod_i H_i^{z_{1,i}} \prod_j \bar{H}(j)^{ce^{j-1}},$$
$$D = g^{z_4} \prod_i X_i^{z_{1,i}} \prod_j \bar{X}(j)^{ce^{j-1}},$$
$$E = h^{z_4} \prod_i Y_i^{z_{1,i}} \prod_j \bar{Y}(j)^{ce^{j-1}}.$$

Hence, we obtain Protocol 12 by applying Protocols 10 and Protocol 11 to Protocol 9. Together with Proposition 2, Theorem 1 and Theorem 2 imply the following.

Corollary 1. *Protocol 12 is an honest verifier of zero-knowledge and is knowledge-sound.*

It suffices to set $m = d$ and $i(j) = j$ to obtain batch processes for proofs of knowledge of R.

Table 3 shows a performance comparison between Protocol 8 and Protocol 12. Our design principle provides advantages over the simple repetition of Protocol 8 with regard to both computation and communications.

Protocol 12
Private input to P **:** $(s(j), t(j), w(j), x(j)) \in (\mathbb{Z}/p\mathbb{Z})^4$
Common input: $i(j) \in \{1, 2, \cdots, m\}$, $(Z(j), \bar{G}(j), \bar{H}(j), \bar{X}(j), \bar{Y}(j))$
$= (g^{x(j)} h'^{w(j)}, g^{s(j)} G_{i(j)}^{x(j)}, h^{s(j)} H_{i(j)}^{x(j)}, g^{t(j)} X_{i(j)}^{x(j)} h^{t(j)} Y_{i(j)}^{x(j)}) \in \mathcal{G}^5$ *,where* $j = 1, \ldots, d$

1. *P chooses* $\delta, \eta_i, \rho, \tau \in_R \mathbb{Z}/p\mathbb{Z}$ *for each* $i = 1, \ldots, m$ *, and* $\alpha(j), \beta(j) \in_R \mathbb{Z}/p\mathbb{Z}$ *for each* $j = 1, \ldots, d$ *randomly. P computes*
$$A = g^{\sum_i \eta_i} h'^{\rho},\ B = g^{\delta} \prod_i G_i^{\eta_i},$$
$$C = h^{\delta} \prod_i H_i^{\eta_i},\ D = g^{\tau} \prod_i X_i^{\eta_i},$$
$$E = h^{\tau} \prod_i Y_i^{\eta_i},$$
$$F(j) = g^{\alpha(j)} G^{\beta(j)} \text{ for each } j = 1, 2, \cdots, d.$$
Then P sends $(A, B, C, D, E, F(1), \cdots, F(d))$ *to V.*
2. *V chooses* $e, c, c(1), c(2), \cdots, c(d) \leftarrow \mathbb{Z}/p\mathbb{Z}$ *randomly.*
3. *Set* $x_i(j) = x(j)$ *for* $i = i(j)$, $x_i(j) = 0$ *for* $i \neq i(j)$. *P computes*
$$z_{1,i} = \eta_i - c \textstyle\sum_j e^{j-1} x_i(j) \bmod p,$$
$$z_2 = \rho - c \textstyle\sum_j e^{j-1} w(j) \bmod p,\ z_3 = \delta - c \textstyle\sum_j e^{j-1} s(j) \bmod p,$$
$$z_4 = \tau - c \textstyle\sum_j e^{j-1} t(j) \bmod p, \quad (i = 1, \ldots, m),$$
$$z_5(j) = \alpha(j) - c(j) s(j) \bmod p,\ z_6(j) = \beta(j) - c(j) x(j) \bmod p,$$
and then sends
$(z_{1,1}, \ldots, z_{1,m}, z_2, z_3, z_4, z_5(1), \cdots, z_5(d), z_6(1), \cdots, z_6(d))$ *to V.*
4. *V accepts if*
$$A = g^{\sum_i z_{1,i}} h'^{z_2} \prod_j Z(j)^{ce^{j-1}}$$
$$B = g^{z_3} \prod_i G_i^{z_{1,i}} \prod_j \bar{G}(j)^{ce^{j-1}},$$
$$C = h^{z_3} \prod_i H_i^{z_{1,i}} \prod_j \bar{H}(j)^{ce^{j-1}},$$
$$D = g^{z_4} \prod_i X_i^{z_{1,i}} \prod_j \bar{X}(j)^{ce^{j-1}},$$
$$E = h^{z_4} \prod_i Y_i^{z_{1,i}} \prod_j \bar{Y}(j)^{ce^{j-1}},$$
$$F(j) = g^{z_5(j)} G_{i(j)}^{z_6(j)} \bar{G}(j)^{c(j)} \text{ for each } j = 1, 2, \cdots, d.$$

6 Conclusion and Future Work

We present a design principle to construct a batch processing protocol for various interactive proofs. For example, the Guillou-Quisquater identification scheme is batched, since it has an additive structure. The principle can be applied to some NP-relations even without additivity. We illustrated a practical application for such a non-additive case, in which both computation and communication are reduced.

However, at this time we have no characterization for interactive proofs such that our batch processing improves its efficiency. For example, [4] presents an honest-verifier ZK protocol for partial knowledge. It seems that our strategy does not apply to such NP-relations, even though an attempt can be made to decompose

it as in Section 5. Characterizing such NP-relations is an open problem where the proposed batch processing can be applied to some decompositions.

Acknowledgment. The authors would like to thank anonymous referees for their useful comments and remarks. The authors added Section 4 on a remark that suggests adding an example of pairing-based protocol.

References

1. M. Bellare, J. Garay, and T. Rabin, "Fast Batch Verification for Modular Exponentiation and Digital Signatures," Advances in Cryptology — EUROCRYPT '98, LNCS 1403, pp. 236–250, Springer-Verlag, 1998.
2. D. Boneh and M. Franklin, "Identity based encryption from the Weil pairing," Advances in Cryptology — CRYPTO 2001, LNCS 2139, pp. 213–229, Springer-Verlag, 2001.
3. E. Brickell, D. Gordon, K. McCurley, and D. Wilson, "Fast exponentiation with precomputation," Advances in Cryptology — EUROCRYPT '92, LNCS 658, pp. 200–207, Springer-Verlag, 1992.
4. R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," Advances in Cryptology — CRYPTO '94, LNCS 839, pp. 174–187, Springer-Verlag, 1994.
5. R. Cramer, "Modular Design of Secure yet Practical Cryptographic Protocols," Ph.D.-thesis, CWI and U. of Amsterdam, 1996.
6. A. Fiat, "Batch RSA," Journal of Cryptology, Vol. 10, pages 75–88, 1997.
7. R. Gennaro, D. Leigh, R. Sundaram, and W. Yerazunis, "Batching Schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices," Advances in Cryptology — ASIACRYPT 2004, LNCS 3329, pp. 276–292, Springer-Verlag, 2004.
8. O. Goldreich, "Foundations of Cryptography," volume I, Cambridge University Press, 2001.
9. L. Guillou and J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," Advances in Cryptology — EUROCRYPT'88, pp. 123–128, Springer-Verlag, 1988.
10. D. M'Raihi, and D. Naccache, "Batch exponentiation: a fast DLP-based signature generation strategy," Proceedings of the 3rd ACM conference on Computer and Communications Security, 1996
11. T.P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," Advances in Cryptology — CRYPTO '91, LNCS 576, pp. 129–140, Springer-Verlag, 1992.
12. C. Schnorr, "Efficient signature generation by smart cards," Journal of Cryptology, Vol. 4, pp. 161–174, 1991.
13. B. Schoenmakers and P. Tuyls, "Practical Two-Party Computation Based on the Conditional Gate," Advances in Cryptology — ASIACRYPT 2004, LNCS 3329, pp. 119–204, Springer-Verlag, 2004.
14. M. Tompa and H. Woll, "Random Self-Reducibility and Zero Knowledge Interactive Proofs of Possession of Information," Proc. of FOCS, pp. 472-482, 1987.

Timing Attacks on NTRUEncrypt Via Variation in the Number of Hash Calls

Joseph H. Silverman and William Whyte

NTRU Cryptosystems, Inc.

Abstract. This report studies timing attacks on NTRUEncrypt based on variation in the number of hash calls made on decryption. The attacks apply to the parameter sets of [8,6]. To mount the attacker, an attacker performs a variable amount of precomputation, then submits a relatively small number of specially constructed ciphertexts for decryption and measures the decryption times. Comparison of the decryption times with the precomputed data allows the attacker to recover the key in greatly reduced time compared to standard attacks on NTRUEncrypt. The precomputed data can be used for all keys generated with a specific parameter set and tradeoffs exist that increase the amount of precomputation in order to decrease the time required to recover an individual key. For parameter sets in [3] that claim k-bit security but are vulnerable to this attack, we find that an attacker can typically recover a single key with about $k/2$ bits of effort.

Finally, we describe a simple means to prevent these attacks by ensuring that all operations take a constant number of SHA calls. The recommended countermeasure does not break interoperability with the parameter sets of [8,6] and has only a slight effect on performance.

1 NTRUEncrypt Overview

In this section we briefly review how NTRUEncrypt works in order to set notation. For further details, see [2,3,6]. Recall that NTRUEncrypt uses the ring of truncated polynomials (also sometime called the ring of convolution polynomials)

$$\mathbf{Z}[X]/(X^N - 1).$$

We denote multiplication in this ring by $*$. At various stages of encryption and decryption the coefficients of these polynomials are reduced modulo q and/or modulo p, where p and q are relatively prime integers. This reduction is always performed so that the reduced coefficients lie in the range from 0 to $p-1$ (respectively 0 to $q-1$). In particular, reduction modulo p and reduction modulo q do not commute with one another. For example,

$$(11 \bmod 7) \bmod 2 = 4 \bmod 2 = 0 \quad \text{and} \quad (11 \bmod 2) \bmod 7 = 1 \bmod 7 = 1.$$

For simplicity in this note, we restrict attention to the case $p = 2$, in which case various polynomials are chosen to be binary (i.e., all coefficients 0 or 1), and in some cases with a fixed number of zeros and ones. To ease notation, we let

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 208–224, 2007.

$$\mathcal{B}_N = \{\text{binary polynomials}\},$$
$$\mathcal{B}_N(d) = \{\text{binary polynomials with exactly } d \text{ ones}\}.$$

An NTRUEncrypt private key consists of a pair of (binary) polynomials f and g. The associated public key is the polynomial

$$\mathsf{h} = p * \mathsf{f}_q^{-1} * \mathsf{g} \bmod q,$$

where f_q^{-1} denotes the inverse of f modulo q. Similarly, we let f_p^{-1} denote the inverse of f modulo p. To speed decryption, the polynomial f is often taken in the form $\mathsf{f} = 1 + p\mathsf{F}$ with $\mathsf{F} \in \mathcal{B}_N(d_F)$, in which case $\mathsf{f}_p^{-1} = 1$. See [3,6] for a discussion. The special form $1 + p\mathsf{F}$ will play an important role in our attack.

Encryption and decryption use two hash functions. We denote them by G and H as in [5]. In practice, they are built using either SHA-1 or SHA-256 in various ways, depending on the desired security level, see [8]. The attack that we describe is based on the fact that the number of SHA calls required by G depends on the input to G. Thus by measuring decryption time, an attacker may obtain information about the input to G, which in turn reveals information about the private key f.

The encryption process works as follows.

$\mathsf{M} \in \mathcal{B}_N$	*Padded plaintext*
$\mathsf{r} = G(\mathsf{M}) \in \mathcal{B}_N(d_r)$	*Randomizer*
$\mathsf{m}' = \mathsf{M} \oplus H(\mathsf{r} * \mathsf{h} \bmod q)$	*Masked message representative*
$\mathsf{e} = (\mathsf{r} * \mathsf{h} + \mathsf{m}') \bmod q$	*Ciphertext*

The decryption algorithm first recovers the (padded) message representative m' and plaintext M and then uses them to recreate the blinding value r and verify that $(\mathsf{m}', \mathsf{e})$ is a valid NTRUEncrypt pair.

$\mathsf{m}' = \big((\mathsf{f} * \mathsf{e} \bmod q) \bmod p\big) * \mathsf{f}_p^{-1} \bmod p$	*Recover candidate* m'
$\mathsf{M} = \mathsf{m}' \oplus H(\mathsf{e} - \mathsf{m}' \bmod q)$	*Unmask* m' *to get* M
$\mathsf{r} = G(\mathsf{M})$	*Recover* r *used in encryption*
Verify that e equals $\mathsf{r} * \mathsf{h} + \mathsf{m}' \bmod q$	

The basis for our timing attack lies in the way in which G uses SHA to create r from M. Note that on decryption, e and m' completely determine M, and therefore the time to calculate $\mathsf{r} = G(\mathsf{M})$. The blinding value r is required to be a binary polynomial with exactly d_r ones, and the process described in [8] for creating r from M may take a different number of SHA calls for different values of M. Later we will describe exactly how this is done, but for now we simply observe that this leads to a time variation that an attacker may be able to measure and show how these timing observations may be converted into information about the private key f. We also note that a simple countermeasure, as described in Section 7, is to perform a few extra SHA calls to ensure that every decryption takes the same amount of time.

2 The Time Trail of a Ciphertext

As we saw in Section 1, the number of hash calls required to create the blinding value r from a message representative/ciphertext pair $(\mathsf{m}', \mathsf{e})$ may be different for different pairs $(\mathsf{m}', \mathsf{e})$. Each hash call requires a nontrivial amount of time, so an adversary might be able to determine how many hash calls Bob uses in attempting to decrypt a (possibly bogus) ciphertext e.

In practice, there will be a number K so that the number of hash calls required to create r from $(\mathsf{m}', \mathsf{e})$ is usually either K or $K+1$. For each pair $(\mathsf{m}', \mathsf{e})$, regardless of whether or not it is a valid NTRUEncrypt pair, we define $\mathsf{r}(\mathsf{m}', \mathsf{e})$ to be the output from the decryption algorithm,

$$\mathsf{r}(\mathsf{m}', \mathsf{e}) = G\big(\big(\mathsf{m}' + H(\mathsf{e} - \mathsf{m}' \bmod q)\big) \bmod 2\big),$$

and we set $\beta(\mathsf{m}', \mathsf{e}) \in \{0, 1\}$ by the rule

$$\beta(\mathsf{m}', \mathsf{e}) = \begin{cases} 0 & \text{if it takes } \le K \text{ hashes to create } \mathsf{r}(\mathsf{m}', \mathsf{e}), \\ 1 & \text{if it takes } > K \text{ hashes to create } \mathsf{r}(\mathsf{m}', \mathsf{e}). \end{cases}$$

Note that for known $(\mathsf{m}', \mathsf{e})$, the computation of $\mathsf{r}(\mathsf{m}', \mathsf{e})$, and thus of $\beta(\mathsf{m}', \mathsf{e})$, requires no private knowledge.

For a given $(\mathsf{m}', \mathsf{e})$, we look also at the rotations $(X^i\mathsf{m}', X^i\mathsf{e})$ for $i = 0, 1, \ldots$. We define the *Time Trail* of $(\mathsf{m}', \mathsf{e})$ to be the binary vector

$$\begin{aligned} T(\mathsf{m}', \mathsf{e}) &= \big(\beta(\mathsf{m}', \mathsf{e}), \beta(X\mathsf{m}', X\mathsf{e}), \beta(X^2\mathsf{m}', X^2\mathsf{e}), \ldots, \beta(X^{N-1}\mathsf{m}', X^{N-1}\mathsf{e})\big) \\ &\in \{0, 1\}^N. \end{aligned}$$

The Time Trail tells us how many hashes are required for each of the rotations of the pair $(\mathsf{m}', \mathsf{e})$.

Let P be the probability that a randomly chosen $(\mathsf{m}', \mathsf{e})$ requires (at most) K hash calls and similarly $1 - P$ is the probability that a randomly chosen $(\mathsf{m}', \mathsf{e})$ requires (at least) $K + 1$ hash calls. If neither P nor $1 - P$ is too small, then the probability that two pairs $(\mathsf{m}'_1, \mathsf{e}_1)$ and $(\mathsf{m}'_2, \mathsf{e}_2)$ have the same time trails is quite small. More precisely, it is not hard to derive the formula

$$\mathrm{Prob}\big(T(\mathsf{m}'_1, \mathsf{e}_1) = T(\mathsf{m}'_2, \mathsf{e}_2)\big) = (1 - 2P + 2P^2)^N.$$

This holds under the assumption that different entries in the vectors are random and independent: this is correct so long as the main variation in running time comes from the hash calls, and so long as the output of SHA-1 is in some sense random. We otherwise defer the derivation of this formula to Section A.1.

3 A Timing Attack Based on Variable Number of Hash Calls

In this section we explain how an adversary Oscar might use time trails in order to derive information about Bob's private key.

Oscar first chooses a collection of (possibly bogus) ciphertexts $\mathcal{E}$ (i.e., $\mathcal{E}$ is a collection of polynomials modulo q). He also chooses a set of message representative values $\mathcal{M}$ (i.e., a collection of binary polynomials) with the property that $\mathcal{M}$ contains many of the polynomials in the set

$$\big\{((\mathsf{f} * \mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2) : \mathsf{e} \in \mathcal{E}\big\}.$$

Note that this is exactly the set of message representative that Bob would create during the process of decrypting the ciphertexts in $\mathcal{E}$. More precisely, we assume that the probability

$$p_{\mathcal{M},\mathcal{E}} := \operatorname*{Prob}_{\mathsf{e}\in\mathcal{E}}\big(((\mathsf{f} * \mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2) \in \mathcal{M}\big)$$

is not too small.

Before starting the active part of the attack, Oscar creates a table consisting of the time trails of every pair in $\mathcal{M} \times \mathcal{E}$. In other words, he creates a searchable list of binary vectors

$$\big(T(\mathsf{m}', \mathsf{e}) : \mathsf{m}' \in \mathcal{M} \text{ and } \mathsf{e} \in \mathcal{E}\big\}.$$

Thus the precomputation required for the attack has time and space requirements that are $O\big(\#\mathcal{M} \cdot \#\mathcal{E}\big)$.

To initiate the attack, Oscar chooses a random $\mathsf{e} \in \mathcal{E}$, sends it to Bob, and records how long it takes Bob to decipher it. Note that the use of NAEP padding [5] as described above ensures that bogus ciphertexts will be rejected. But in this case the attacker does not care that the ciphertexts are rejected, so long as he can obtain timing information. This timing information enables him to determine how many hash calls are required to create r from the ciphertext e and the message representative

$$\mathsf{m}'(\mathsf{e}) := ((\mathsf{f} * \mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2),$$

so Oscar finds the value of $\beta(\mathsf{m}'(\mathsf{e}), \mathsf{e})$. Of course, Oscar does not know the value of $\mathsf{m}'(\mathsf{e})$.

In a similar manner, Oscar sends each of the polynomials

$$\mathsf{e}, \quad X\mathsf{e}, \quad X^2\mathsf{e}, \quad X^3\mathsf{e}, \quad \ldots, \quad X^{N-1}\mathsf{e}$$

to Bob and obtains the values $\beta(\mathsf{m}'(X^i\mathsf{e}), X^i\mathsf{e})$ for $i = 0, 1, \ldots, N-1$. We now observe that

$$\begin{aligned}\mathsf{m}'(X^i\mathsf{e}) &= ((\mathsf{f} * X^i\mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2)\\ &= X^i * ((\mathsf{f} * \mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2)\\ &= X^i\mathsf{m}'(\mathsf{e})\end{aligned}$$

Thus Oscar has determined $\beta(X^i\mathsf{m}'(\mathsf{e}), X^i\mathsf{e})$ for $i = 0, 1, \ldots, N-1$, so he knows the time trail $T(\mathsf{m}'(\mathsf{e}), \mathsf{e})$ of the pair $(\mathsf{m}'(\mathsf{e}), \mathsf{e})$.

Oscar now searches his precomputed list and, with reasonable probability, finds a small number of possibilities for $(\mathsf{m}'(\mathsf{e}), \mathsf{e})$. In other words, Oscar now has a known polynomial e and a known polynomial m' so that when Bob decrypted e, Bob got m' as the message representative. Hence Oscar knows that there is an equation of the form

$$\mathsf{m}' * \mathsf{f} \equiv (\mathsf{f} * \mathsf{e} \bmod q) \pmod 2. \tag{1}$$

(More precisely, Oscar knows e and he has a small list of possible m', one of which satisfies (1). In Section 4.1 we discuss how Oscar can disambiguate between the possible m' in a plausible attack scenario.) Equation (1) certainly contains a significant amount of information concerning Bob's private key f, although exploiting this information will depend on the specific form of e. For example, if the elements of $\mathcal{E}$ consist of polynomials with very few nonzero coefficients, then equation (1) may give information concerning the spacing between the nonzero coefficients of f. In Section 4 we describe a specific collection $\mathcal{E}$ that leads to a practical hash timing attack when the key f has the form $\mathsf{f} = 1 + p\mathsf{F}$. (This form is sometimes used to decrease decryption time.)

4 A Practical Hash Timing Attack for $\mathsf{f} = 1 + 2\boldsymbol{F}$ — Theory

For this section we consider the case where $p = 2$, so q is necessarily odd, and where private keys have the form

$$\mathsf{f} = 1 + 2\mathsf{F} \qquad \text{for some binary polynomial } \mathsf{F} \in \mathcal{B}_N(d_F).$$

The parameters recommended by NTRU Cryptosystems currently take this form [3,6,8] Note that the inverse $\mathsf{f}_2^{-1} = (\mathsf{f} \bmod 2)^{-1}$ is equal to 1, so the formula that Bob uses to recover the message representative m' from a ciphertext e simplifies to

$$\mathsf{m}'(\mathsf{e}) = (\mathsf{f} * \mathsf{e} \bmod q) \bmod 2. \tag{2}$$

For later computations, we write $\mathsf{F} = \sum_j F_i X^j$ with $F_j \in \{0, 1\}$, and for any $j \in \mathbf{Z}$, we let F_j denote the coefficient $F_{(j \bmod N)}$.

Let $\lambda = 2\lceil q/8 \rceil$ be the smallest even integer that is larger than $q/4$. To mount the attack, Oscar uses the set of (bogus) ciphertexts defined by

$$\mathcal{E} = \bigl\{\lambda + \lambda X^i : 1 \le i < N\bigr\}.$$

In other words, the $\mathsf{e} \in \mathcal{E}$ are polynomials with two coefficients equal to λ and all other coefficients equal to 0. In summary, Oscar's attack is:

1. Choose a value δ.
2. Let $\mathcal{E} = \{\mathsf{e}_i = \lambda + \lambda X^i : 0 \le i \le (N-1)/2\}$ and $\mathcal{M} = \mathcal{B}_N(0 < d \le \delta)$.
3. Precompute and store in a suitably searchable database the time trails $T(\mathsf{m}', \mathsf{e})$ for every $\mathsf{m}' \in \mathcal{M}$ and every $\mathsf{e} \in \mathcal{E}$.

4. For each i, send e_i, $X\mathsf{e}_i, \ldots, X^{N-1}\mathsf{e}_i$ to Bob and use the decryption times to determine the time trail $T(\mathsf{m}'(\mathsf{e}_i), \mathsf{e}_i)$ as described in Section 3.
5. Search the database to determine $\mathsf{m}'(\mathsf{e}_i)$, either exactly or up to a small number of choices. Once a candidate $\mathsf{m}'(\mathsf{e}_i)$ is found, validate it by the methods below.
6. Use the resulting values of $\mathsf{m}'(\mathsf{e}_i)$ to reconstruct F, either by an exact computation or by cutting down on the search space for F and performing a direct search of that subset.

We now need to figure out the possible values of $\mathsf{m}'(\mathsf{e})$ that arise in (2) when Bob decrypts the ciphertexts in $\mathcal{E}$. During decryption, Bob first computes

$$\begin{aligned} a &= \mathsf{f} * \mathsf{e} \bmod q \\ &\equiv (1+2F) * (\lambda + \lambda X^i) \pmod q \\ &\equiv \lambda + \lambda X^i + \sum_{j=0}^{N-1} 2\lambda(F_j + F_{j-i})X^j. \end{aligned}$$

Thus the j^{th} coefficient of a is given by

$$a_j = \begin{cases} \lambda(1 + 2F_0 + 2F_{-i}) \bmod q & \text{if } j = 0, \\ \lambda(1 + 2F_i + 2F_0) \bmod q & \text{if } j = i, \\ \lambda(2F_j + 2F_{j-i}) \bmod q & \text{if } j \neq 0, i \end{cases} \tag{2}$$

The key observation is that since $\lambda = 2\lceil q/8 \rceil$ is just slightly larger than $q/4$, the quantities on the righthand side of 2 are between 0 and $q-1$ unless $F_j = F_{j-i} = 1$, in which case they are greater than q. Thus there is nontrivial reduction modulo q if and only if $F_j = F_{j-i} = 1$, which implies that

$$a_j = \begin{cases} \lambda,\ 2\lambda,\ \text{or } 3\lambda & \text{if } F_j = 0 \text{ or } F_{j-i} = 0, \\ 4\lambda - q \text{ or } 5\lambda - q & \text{if } F_j = F_{j-i} = 1. \end{cases}$$

The next step is to reduce a modulo 2, which yields the message representative $\mathsf{m}'(\mathsf{e}_i)$ for the (bogus) ciphertext $\mathsf{e}_i = \lambda + \lambda X^i$. Recalling that λ is even and q is odd, we see that

$$a_j \bmod 2 = \begin{cases} 0 & \text{if } F_j = 0 \text{ or } F_{j-i} = 0, \\ 1 & \text{if } F_j = F_{j-i} = 1. \end{cases}$$

This gives the following explicit description of $\mathsf{m}'(\mathsf{e}_i)$:

$$\mathsf{m}'(\mathsf{e}_i) = \sum_{j=0}^{N-1} \left(\begin{array}{l} 1 \text{ if } F_j = F_{j-i} = 1 \\ 0 \text{ otherwise} \end{array} \right) X^j \ ,$$

which in turn yields the following partial information about F:

$$\mathsf{F}(\mathsf{e}_i) = \sum_{j=0}^{N-1} \begin{pmatrix} 1 \text{ if } \mathsf{m}'(\mathsf{e}_i)_j = 1 \\ \quad \text{or } \mathsf{m}'(\mathsf{e}_i)_{j+i} = 1 \\ 0 \text{ if } \mathsf{m}'(\mathsf{e}_i)_{j-i} = 1 \text{ and } \mathsf{m}'(\mathsf{e}_i)_j \neq 1 \\ \quad \text{or } \mathsf{m}'(\mathsf{e}_i)_{j+2i} = 1 \text{ and } \mathsf{m}'(\mathsf{e}_i)_{j+i} \neq 1 \\ ? \text{ unknown otherwise} \end{pmatrix} X^j \ ,$$

Therefore, every m' with $d_{m'}$ ones that Oscar can recover will yield $d_{m'}$ pairs of non-zero coefficients of F, allowing him to reduce the search space for F. To be precise, defining the "left-hand" member of a pair in the obvious way, we see that each of the $d_{m'}$ left-hand members must be distinct, at least one of the right-hand members must not occupy the same location as the left-hand member of another pair (because N is prime) and for the remaining $d_{m'} - 1$ right-hand members the expected number of left-hand members that they occupy the same position as is given by the expected value of the hypergeometric distribution,

$$\frac{(d_{m'} - 1)^2}{N - d_{m'} - 2}$$

The expected number of distinct coefficients of value 1, $c_1(d_{m'}, N)$, is therefore

$$c_1(d_{m'}, N) = 2d_{m'} - \frac{(d_{m'} - 1)^2}{N - d_{m'} - 2}$$

Oscar will also have learned the location of some of the zero coefficients of F: each 1 coefficient that is not known to have another 1 i places to its left or to its right must have a zero in that position (as a 1 would have been detected, and the only other option is 0). This, too, will allow him to reduce the search space for F.

Now we estimate the amount of precomputation that Oscar must carry out in order to mount the attack.

First, we note that the running time of a standard combinatorial attack on an NTRUEncrypt private key is [4]

$$\tau(d_F, N) = \frac{1}{\sqrt{N}} \binom{\lceil N/2 \rceil}{\lceil d_F/2 \rceil}$$

If Oscar knows the locations of d_1 1s and d_0 0s in F, this running time becomes

$$\tau(d_F, N; d_0, d_1) = \binom{N - (d_0 + d_1 + \lfloor (d_F - d_1)/2 \rfloor)}{\lceil \frac{d_F - d_1}{2} \rceil} \tag{3}$$

(the top line here is about N, rather than about $N/2$, because rotational symmetry has been broken, and the factor of $1/\sqrt{N}$ vanishes for the same reason. Both of these changes hinder the attacker). Oscar's aim is to balance precomputation work and key-specific combinatorial work so as to recover a key in as little total effort as possible.

Oscar will start by selecting an integer δ and precomputing the time trails for all m' such that $d_{m'} \leq \delta$. We now estimate how many coefficients of F this

will enable him to recover. For any $(d_{m'}, i)$, we want to calculate the probability that F has exactly $d_{m'}$ pairs of coefficients separated by i, or in other words the probability that the dot product $(\mathsf{F} \cdot X^i \mathsf{F}) = d_{m'}$. To estimate this, consider what would happen if F and $X^i\mathsf{F}$ were independent. Each of the d_F 1 coefficients in F will select one of the coefficients in X^iF, giving a hypergeometric distribution of the values of $(\mathsf{F} \cdot X^i\mathsf{F})$. In practice, we know that if $i \neq 0$, a given 1 in F cannot select itself in X^iF and we observe that

$$P_{N,d_F}[\mathsf{F} \cdot X^i F = d_{m'}] = \mathrm{Hyp}(d_{m'}, d_F - 1, d_F, N) = \frac{\binom{d_F}{d_{m'}}\binom{N-d_F}{d_F-1-d_{m'}}}{\binom{N}{d_F-1}}$$

where $\mathrm{Hyp}(x, d, s, N)$, the hypergeometric distribution, is the probability of x successes in d draws without replacement from a pool containing N items of which s count as successes.

For any given value of $d_{m'}$, there are $(N-1)/2$ different values of i and $(N-1)/2$ distinct e_i. The expected number of m's with $d_{m'}$ 1s is therefore

$$E_1(d_{m'}) = \frac{(N-1)}{2} * \mathrm{Hyp}(d_{m'}, d_F - 1, d_F, N)$$

and the amount of precomputation work required to generate these time trails is

$$w_N(d_{m'}) = \frac{N(N-1)}{2}\binom{N}{d_{m'}}.$$

Every successful time trail identifies c_1 distinct 1 coefficients,

$$c_1(d_{m'}, N) = 2d_{m'} - \frac{(d_{m'}-1)^2}{N - d_{m'} - 2}.$$

It also identifies c_0 distinct 0s, one to the left of every lefthand 1 and one to the right of every righthand 1 except for the 1s that are lefthand in one pair and righthand in another:

$$c_1(d_{m'}, N) = 2d_{m'} - 2\frac{(d_{m'}-1)^2}{N - d_{m'} - 2}.$$

We now consider how quickly Oscar learns the distinct coefficients of F. Say that he knows d_0 0s and d_1 1s, and as a result of finding a time trail he discovers an additional c_0 0s and c_1 1s. Then we estimate the new expected total number of distinct known coefficients as

$$\begin{aligned}
\text{(new total)} &= \text{(already known)} + \text{(new)} - \text{(collisions between old and new)} \\
d_1' &= d_1 + c_1 - \mathrm{Exp}_x(\mathrm{Hyp}(x, c_1, d_1, d_F)) \\
&= d_1 + c_1 - \frac{d_1 c_1}{d_F} \\
d_0' &= d_0 + c_0 - \mathrm{Exp}_x(\mathrm{Hyp}(x, c_0, d_0, N - d_F)) \\
&= d_0 + c_0 - \frac{d_0 c_0}{N - d_F}.
\end{aligned}$$

This allows us to calculate the expected number of distinct coefficients found for a certain amount of precomputation corresponding to a certain value of δ, and therefore estimate the amount of work left to be done to recover the key. The method is:

1. Set $d_0 = d_1 = 0$. Set the total work $w = 0$.
2. For $d_{m'} = 1$ to δ:
3. Calculate $E_1(d_{m'})$.
4. If $\lfloor E_1(d_{m'}) \rceil \geq 1$:
   (a) Calculate $c_1(d_{m'}, N)$, $c_0(d_{m'}, N)$.
   (b) For $i = 1$ to $\lfloor E_1(d_{m'}) \rceil$:
   (c) Set $d_1 = d_1 + c_1 - \frac{d_1 c_1}{d_F}$.
   (d) Set $d_0 = d_0 + c_0 - \frac{d_0 c_0}{N - d_F}$
   (e) End i loop.
5. Set $w = w + w_N(d_{m'})$
6. End $d_{m'}$ loop.
7. Calculate $\tau(d_F, N; d_0, d_1)$ by (3) and output w, τ.

We emphasise that this is simply an estimate, and in particular the use of the hypergeometric distribution is an approximation to the actual distribution. The aim is simply to motivate a choice for δ.

4.1 Validating a Choice

The initial set of (bogus) ciphertexts $\mathcal{E} = \{\mathsf{e}_i = \lambda + \lambda X^i\}$ is relatively small to reduce precomputation. Recognizing a time trail will tell Oscar that with high probability he has identified $\mathsf{m}'(\mathsf{e}_i)$ for the relevant e_i in his database. However, if there is a nontrivial chance that the time trail is nonunique, Oscar may want to check that he has in fact identified the correct e_i.

To see how to do this, we note that if

$$\mathsf{e}_i = \lambda + \lambda X^i$$

decrypts to m', then so do the alternate forms

$$\mathsf{e}_i^* = (\lambda + 2) + \lambda X^i, \quad \text{or} \quad \lambda + (\lambda + 2)X^i, \quad \text{or} \quad \ldots$$

or indeed many polynomials $\lambda_1 + \lambda_2 X^i$ with λ_1 and λ_2 even integers in the vicinity of $q/4$ and satisfying $\lambda_1 + \lambda_2 > q/2$. Oscar therefore selects one of the possible e_i^*, calculates the time trail $T(\mathsf{m}', \mathsf{e}_i^*)$ for the message representative m' that he thinks is produced by decrypting the original e_i, and then submits e_i^* to the decryption oracle to find its time trail. If the measured and the calculated time trail match, he has confirmed the guess for m'. Otherwise, he knows that the original match on the time trail was just coincidence.

4.2 Results

We present the results of our analysis in Table 1. Here we have calculated two different values of δ.

The value δ_{onekey} is the value of δ that Oscar will precompute up to if he wants to recover one key, in other words the first value of δ for which the work required to peform the precomputation up to δ, $w(\delta)$, is greater than the remaining work required to break the key, τ. It can be seen that, with the exception of parameter set `ees251ep4` (presented in [7]), the log of the amount of precomputation to be performed $\log_2 w(\delta)$ is slightly more than half the claimed bit strength of the parameter sets. We also present, for interest, the number of distinct 1s and 0s that Oscar will on average have identified in a target key before he starts the combinatorial attack on the remaining coefficients.

The value δ_{allkeys} is the value of δ at which Oscar will on average recover the locations of all d_F value-1 coefficients in F through time trail analysis alone. This is the amount of precomputation that will allow him to recover any key at the cost of simply submitting about $N(N-1)$ ciphertexts for decryption. It can be seen that in general $w(\delta_{\text{allkeys}})$ is greater than $w(\delta_{\text{onekey}})$ by about 11 bits, or a factor of about 2000.

This demonstrates that, so long as the time trails are sufficiently unique and Oscar has an amount of storage that is customarily granted to attackers in this kind of paper, this attack is practical. In the next section we analyse the probability that time trails are unique.

Table 1. precomputation effort required to recover one key with minimum work and to recover all keys for the parameter sets in [7,8]

Bit Security	Parameter Set Name	N	d_r, d_F	$\delta_{\text{one key}}$	c_1	c_0	w	τ	$\delta_{\text{all keys}}$	w
80	`ees251ep4`	251	72	14	51.31	57.35	89.75	51.66	16	97.62
80	`ees251ep6`	251	48	5	40.26	64.72	47.86	23.99	7	58.36
112	`ees347ep3`	347	66	7	50.52	72.97	62.59	46.58	9	73.24
128	`ees397ep1`	397	74	8	60.81	94.48	69.95	42.74	10	80.67
160	`ees491ep1`	491	91	10	71.49	105.17	84.38	60.54	12	95.16
192	`ees587ep1`	587	108	12	79.57	110.07	98.79	88.09	14	109.62
256	`ees787ep1`	787	140	16	112.70	169.35	127.72	88.58	18	138.67

5 A Practical Hash Timing Attack for $\mathbf{f} = 1 + 2\boldsymbol{F}$ — Practice

In this section we evaluate the practicality of the attack described in Section 4 for some specific NTRUEncrypt parameter sets that appear in [6,8] (and also the parameter set `ees251ep4` described in [7], which is secure but less efficient than the corresponding parameter sets in [8]). This practicality depends, among other

things, on the probability that different inputs require a greater or lesser number of SHA calls. We begin by describing how [8] uses SHA to compute r and then we compute the probability that this process takes a varying number of SHA calls.

The blinding value r, which is a binary polynomial with exactly d_r ones, is created from a hash function via repeated calls to some version of SHA. Here is the process as described in [8]:

1. Fix a value of c satisfying $2^c > N$. This value of c is specified in [8] for each of the sample NTRUEncrypt parameter sets. Also let
$$b = \lceil c/8 \rceil \qquad \text{and} \qquad n = \lfloor 2^c/N \rfloor$$
Thus b is the smallest integer such that b bytes contains at least c bits. (In practice, b will be 1 or 2.) Similarly, nN is the smallest multiple of N that is less than 2^c.
2. Call the specified version of SHA and break the output into chunks of b bytes each. Within each b byte chunk, keep the lower order c bits and discard the upper order $8b - c$ bits. Convert the lower order c bits into (little endian) integers $i_1, i_2, \ldots, i_t$. (Here t is the integer such that the output of the specified version of SHA consist of tb bytes.) This process of splitting the output from SHA is illustrated in Figure 1.
3. Create a list of indices $j_1, j_2, \ldots$ by looping through the list of i values from (2). If $i < n$ and i mod N is not already in the list, the adjoin i mod N to the list, otherwise discard i. Continue until the list contains d_r values of j. If at any point you run out of i values, then call SHA and create additional i values as specified in (2). The complete r generation algorithm is illustrated with pseudocode in Figure 2.

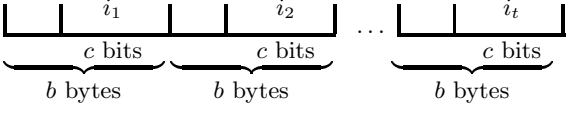

Fig. 1. Converting SHA output into c bit integers

```
(1)  jList = { }
(2)  Call SHA to get i_1, i_2, ..., i_t
(3)  Loop α = 1, 2, ..., t
(4)       If i_α < n and (i_α mod N) ∉ jList
              then adjoin i_α mod N to jList
(5)       If jList contains d_r elements, then exit
(6)  End α loop
(7)  Go to Step (2) to get more i values
```

Fig. 2. Generating r from SHA output

This description makes it clear why the number of calls to SHA may vary for different input values. If we treat the list of numbers $i_1, i_2, \ldots$ as a random sequence of integers in the range $0 \le i < 2^c$, the fundamental probabilities that we need to compute are

$$P_{C,N,n}(L,d) = \text{Prob}\left(\begin{array}{l}\text{A set of } L \text{ randomly chosen integers } i \in [0,C)\\ \text{includes exactly } d \text{ numbers satisfying both}\\ i \in [0,nN) \text{ and the values are distinct modulo } N\end{array}\right)$$

It is not hard to find a recursive formula that allows one to compute $P_{C,N,n}(L,d)$ reasonably quickly. See Appendix C for details.

In order to generate r, the algorithm described in Figure 2 needs to create a list of d_r distinct numbers satisfying $0 \le i < N$. Each time the algorithm calls SHA, it gets t numbers satisfying $0 \le i < 2^c$. Hence the probability that it suffices to call to SHA s times is equal to the probability that st random numbers in the range $[0, 2^c)$ contain at least d_r values in $[0, n)$ whose values modulo N are distinct. Hence

$$\text{Prob}(s \text{ calls to SHA suffices}) = \text{Prob}\left(\begin{array}{l}st \text{ randomly chosen integers in } [0,2^c)\\ \text{includes } \underline{\text{at least}}\ d_r \text{ values in } [0,n)\\ \text{that are distinct modulo } N\end{array}\right)$$

$$= \sum_{d_r \le d \le st} P_{2^c,N,n}(st,d).$$

In Table 2 we have assembled the NTRUEncrypt parameters from [7,8] and computed the values of s such that it is most likely to take either s or $s+1$ calls to SHA in order to generate r. The probabilities are listed in the last column of the table. The closer that the first probability is to 50%, the greater the chance that a time trail is unique, reducing the need to validate a time trail using the methods of Section 4.1. In most cases except perhaps $k = 80$ and $k = 192$, it will not be necessary to validate a time trail.

Table 2. The probability that s calls to SHA generates r

Bit Security	N	d_r	SHA bits	c	b	n	t	s : Prob(s SHA calls suffices)		$P_{\text{nonunique}}$
80	251	48	160	8	1	1	20	3 : 98.14%	4 : 100.0%	$2^{-13.5}$
112	347	66	160	14	2	47	10	7 : 15.65%	8 : 98.48%	2^{-154}
128	397	74	160	11	2	5	10	8 : 12.77%	9 : 95.10%	2^{-144}
160	491	91	160	9	2	1	10	10 : 13.87%	11 : 91.32%	2^{-193}
192	587	108	256	11	2	3	16	8 : 4.52%	9 : 82.38%	2^{-76}
256	787	140	256	12	2	5	16	10 : 53.04%	11 : 99.85%	2^{-783}

6 Practicality of Attack: Availability of Timing Information

As noted, it is possible for decryption to take a variable amount of time depending on the number of hash calls made. In this section we investigate how likely it is that this information will be leaked.

On a 1.7 GHz Pentium Pro running Windows XP, NTRUEncrypt decryption with the `ees251ep6` parameter set of [8] takes 0.09 ms. A SHA-1 call takes about 1.34μs. These are average figures. The time for these averages to settle down is obviously of interest.

We ran 100 sets of experiments, in each of which we decrypted a given `ees251ep6` ciphertext 1,000,000 times. As expected from Table 2, 98 of these ciphertexts took 3 SHA-1 calls to generate r and the other 2 took 4. We sorted the 100 experiments by running time and hoped to see that the 2 cases where there had been 4 SHA-1 calls would also have the longest running times. In fact, the noise due to other system activity overwhelms the variation in running time due to the number of hash calls on this system: the two cases where there had been 4 SHA-1 calls were in 29th and 68th position on the sorted list. Each of these runs took about 90 seconds. If the noise could be eliminated by bombarding the decryption oracle with the same ciphertext for a period of an hour, it would take the attacker $N(N-1)/2$ hours to recover all the time trails, or approximately $3\frac{1}{2}$ years. It therefore appears that this attack is unlikely to succeed against an implementation of NTRUEncryptdecryption running on a general computing platform.

At the other end of the computing scale, on an 8051-type smart card (a Philips Mifare ProX running at a 2.66 MHz internal clock, simulated on the Keil tools simulator) we observed that for `ees251ep4` the total time for a decryption was 58 ms, of which 30 ms was due to the 6 SHA-1 calls. In other words, on this platform, an additional SHA call incurs an overhead of 5 ms. It seems highly likely that in this environment the attack described in this paper is practical.

7 Conclusions and Recommendations

We have described a timing attack on the implementation of NTRUEncrypt described in [8]. The attack relies on the fact that decryption of different (possibly bogus) ciphertexts may require a different number of calls to a hash function such as SHA-1 or SHA-256. We draw some conclusions and make some recommendations.

1. The attack appears unlikely to work against NTRUEncryptrunning on a general-purpose PC platform. However, the parameter sets of [8] are claimed to be appropriate for any platform and as such it is worth investigating countermeasures that can be put in place on any platform.
2. Although we have only described an attack that relies on keys of the special form $\mathsf{f} = 1 + p\mathsf{F}$, it is reasonable to assume that similar attacks are possible for more general keys. Thus the use of general keys is not a recommended method to thwart hash timing attacks on NTRUEncrypt.
3. In order to prevent hash timing attacks, it suffices to make sure that almost all decryptions require the same number of SHA calls. This can be accomplished by fixing a parameter K_{SHA} so that almost all inputs $(\mathsf{m}', \mathsf{e})$ require at most K_{SHA} SHA calls and then performing extra SHA call(s) if necessary so that almost all inputs require exactly K_{SHA} SHA calls. Here, we can

put a more concrete meaning on "almost all" by requiring that at the k-bit security level, there is a chance of 2^{-k} that a given $(\mathsf{m}', \mathsf{e})$ has $\beta(\mathsf{m}', \mathsf{e}) = 1$. This yields the values given in Table 3 for K_{SHA}. Note that even with this number of SHA calls it is expected that decryption will take less than 0.5 s on the smartcard platform described above.

Table 3. Recommended number of SHA calls for different security levels

Bit Security	N	Expected SHA calls	K_{SHA}
80	251	3	6
112	347	8	15
128	397	9	17
160	491	11	22
192	587	9	20
256	787	10	21

Note that this recommendation will require an attacker to expend more than 2^k machine cycles to mount the attack, first because a SHA call takes more than one operation, and second because each attack involves $K_{\mathrm{SHA}} > 1$ SHA calls.

4. The method used to generate r from $(\mathsf{m}', \mathsf{e})$ in [8] is easy to implement, but it is somewhat wasteful of the pseudorandom bits produced by SHA. It might be worthwhile to look for more efficient ways to generate r which might also use a fixed number of calls to SHA, thereby eliminating the possibility of a hash timing attack. However, we note that the use of a new r-generation method would require changes to the exisiting standards, while equalization of the number of SHA calls as in (2) is a simple implementation change that maintains current standards.

   Finally, we note that NTRUEncryptshould continue to be analysed for its vulnerability to other side-channel attacks: this paper is by no means the last word on the subject.

References

1. D. Brumley, D. Boneh, Remote timing attacks are practical. Journal of Computer Networks, 2005.
2. J. Hoffstein, J. Pipher, J.H. Silverman, NTRU: A new high speed public key cryptosystem, Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, Lecture Notes in Computer Science 1423, J.P. Buhler (ed.), Springer-Verlag, Berlin, 1998, 267–288
3. J. Hoffstein, J.H. Silverman, Optimizations for NTRU, Public Key Cryptography and Computational Number Theory (Warsaw, Sept. 11–15, 2000), Walter de Gruyter, Berlin–New York, 2001, 77–88.
4. N. A. Howgrave-Graham, J. H. Silverman, W. Whyte, A Meet-in-the-Middle Attack on an NTRU Private key, Technical report, NTRU Cryptosystems, June 2003. Report #004, version 2, available at `http://www.ntru.com`.

5. N. Howgrave-Graham, J. H. Silverman, A. Singer and W. Whyte. *NAEP: Provable Security in the Presence of Decryption Failures*, IACR ePrint Archive, Report 2003-172, `http://eprint.iacr.org/2003/172/`
6. N. Howgrave-Graham, J. H. Silverman, W. Whyte Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3, Topics in cryptology—CT-RSA 2005, 118–135, Lecture Notes in Comput. Sci., 3376, Springer, Berlin, 2005. `www.ntru.com/cryptolab/articles.htm#2005_1`
7. Consortium for Efficient Embedded Security, *Efficient Embedded Security Standard (EESS) #1 version 2*, 2003.
8. Consortium for Efficient Embedded Security, *Efficient Embedded Security Standard (EESS) #1 version 3*, 2005.

A Probability That Two Message Representatives Have the Same Time Trail

A time trail is a binary vector of dimension N. We let P denote the probabilty that a randomly chosen coordinate is equal to 0, so $1-P$ is the corresponding probablity that a randomly chosen coordinate is equal to 1. Then the probability that (say) the first coordinates of two random time trails agree is

$$\text{Prob(both 0)} + \text{Prob(both 1)} = P^2 + (1-P)^2 = 1 - 2P + 2P^2.$$

In order for two entire time trails to be identical, they must agree on all N of their coordinates. Hence

$$\text{Probability that two Time Trails coincide} = (1 - 2P + 2P^2)^N.$$

Therefore for any given $\mathsf{e} \in \mathcal{E}$ and $\mathsf{m}' \in \mathcal{M}$, the probability that there exists some other message representative $\mathsf{m}'' \in \mathcal{M}$ with $T(\mathsf{e}, \mathsf{m}'') = T(\mathsf{e}, \mathsf{m}')$ is approximately

$$\#\mathcal{M} \cdot (1 - 2P + 2P^2)^N.$$

B The Average Number of Ones with a Given Separation Distance

Let $\mathcal{B}_N(d)$ be the set of binary polynomials of degree less than N with exactly d ones and $N-d$ zeros. Fix i. We are interested in the average number of j such that F_j and F_{j-i} are both equal to 1, as F ranges over $\mathcal{B}_N(d)$. For a given F and i, we denote the number of such j by

$$\nu_i(\mathsf{F}) = \#\{0 \le j < N : F_j = F_{j-i} = 1\}.$$

Clearly $\nu_0(\mathsf{F}) = d$ for every $\mathsf{F} \in \mathcal{B}_N(d)$. We now fix some $1 \le i < N$ and compute the average value $\bar{\nu}_i(d)$ of $\nu_i(\mathsf{F})$ as F ranges over $\mathcal{B}_N(d)$.

$$\begin{aligned}
\bar{\nu}_i(d) = \operatorname*{Average}_{\mathsf{F}\in\mathcal{B}_N(d)} \quad \nu_i(\mathsf{F}) &= \binom{N}{d}^{-1} \sum_{\mathsf{F}\in\mathcal{B}_N(d)} \nu_i(\mathsf{F}) \\
&= \binom{N}{d}^{-1} \sum_{\mathsf{F}\in\mathcal{B}_N(d)} \sum_{j=0}^{N-1} F_j F_{j-i} \\
&= \binom{N}{d}^{-1} \sum_{j=0}^{N-1} \sum_{\mathsf{F}\in\mathcal{B}_N(d)} F_j F_{j-i} \\
&= \binom{N}{d}^{-1} \sum_{j=0}^{N-1} \#\big\{\mathsf{F}\in\mathcal{B}_N(d) : F_j = F_{j-i} = 1\big\} \\
&= \binom{N}{d}^{-1} \sum_{j=0}^{N-1} \binom{N-2}{d-2} \\
&= \binom{N}{d}^{-1} N\binom{N-2}{d-2} \\
&= \frac{d(d-1)}{N}.
\end{aligned}$$

This proves the formula cited in Section 4.

We also observe that $\nu_i(\mathsf{F})$ appears as a coefficient of the product $\mathsf{F} * \mathsf{F}^{\mathrm{rev}}$, where the reversal $\mathsf{F}^{\mathrm{rev}}$ of F is the polynomial $\mathsf{F}^{\mathrm{rev}} = \sum F_{-i}X^i$. Thus

$$\mathsf{F} * \mathsf{F}^{\mathrm{rev}} = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} F_j F_{-k} X^{j+k} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} F_j F_{j-i} X^i = \sum_{i=0}^{N-1} \nu_i(\mathsf{F}) X^i.$$

Thus knowledge of $\nu_i(\mathsf{F})$ for $0 \le i < N$ is equivalent to knowledge of the product $\mathsf{F} * \mathsf{F}^{\mathrm{rev}}$. Using this value and the public key $\mathsf{h} = \mathsf{f}^{-1} * \mathsf{g} \bmod q$, there are practical methods for recovering F. In any case, it is certainly true that each valid $(\mathsf{r}, \mathsf{m}')$ pair that Oscar finds contains significant information about the private key f, and there are numerous ways to exploit such information in order to recover f directly (if one has enough $(\mathsf{r}, \mathsf{m}')$ pairs) or by cutting down the search space for f.

C The Probability of Choosing Distinct Values in a Given Range

In this section we describe a recursion that can be used to compute the probability

$$P_{C,N,n}(L,d) = \mathrm{Prob}\left(\begin{array}{l}\text{A set of } L \text{ randomly chosen integers } i \in [0, C) \\ \text{includes exactly } d \text{ numbers satisfying} \\ i \in [0, nN) \text{ and whose values are distinct modulo } N\end{array}\right)$$

We obtain a recursion from the observation that $P_{C,N,n}(L,d)$ equals the sum of the following two quantities:

- The probability after $L-1$ picks of having $d-1$ values in $[0,nN)$ that are distinct modulo N multiplied by the probability of picking an integer in $[0,nN)$ multiplied by the probability that it does not a repeat a previous values modulo N.
- The probability after $L-1$ picks of having d values in $[0,nN)$ that are distinct modulo N multiplied by the probability of picking an integer that either is not in $[0,nN)$ or whose value modulo N repeats a previous value.

We observe that for the first case, the probability of picking an integer in $[0,nN)$ multiplied by the probability that it does not a repeat a previous values modulo N is

$$\frac{nN}{C}\cdot\frac{N-(d-1)}{N}=\frac{n(N-d+1)}{C}.$$

For the second case, there are $C-nN$ integers in $[0,C)$ that are not in $[0,nN)$, and there are nd integers in $[0,nN)$ that are in one of the d congruence classes modulo n that have already been selected, so the probability of picking an integer that either is not in $[0,nN)$ or whose value modulo N repeats a previous value is

$$\frac{C-nN+nd}{C}=1-\frac{n(N-d)}{C}.$$

This yields the recursion formula

$$\begin{aligned}P_{C,N,n}(L,d)=P_{C,N,n}(L-1,d-1)\cdot\left(\frac{n(N-d+1)}{C}\right)\\ +P_{C,N,n}(L-1,d)\cdot\left(1-\frac{n(N-d)}{C}\right)\end{aligned}$$

Combining this recursion with the obvious initial values

$$P_{C,N,n}(L,d)=0\quad\text{if } L<d\qquad\text{and}\qquad P_{C,N,n}(L,0)=\left(1-\frac{nN}{C}\right)^{L},$$

it is an easy matter to compute $P_{C,N,n}(L,d)$ if the parameters are not too large.

Predicting Secret Keys Via Branch Prediction

Onur Acıiçmez[1], Çetin Kaya Koç[1,2], and Jean-Pierre Seifert[3,4]

[1] School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331, USA
[2] Information Security Research Center
Istanbul Commerce University
Eminönü, Istanbul 34112, Turkey
[3] Applied Security Research Group
The Center for Computational Mathematics and Scientific Computation
Faculty of Science and Science Education
University of Haifa
Haifa 31905, Israel
[4] Institute for Computer Science
University of Innsbruck
6020 Innsbruck, Austria
aciicmez@eecs.oregonstate.edu, koc@cryptocode.net,
jeanpierreseifert@yahoo.com

Abstract. This paper announces a new software side-channel attack — enabled by the branch prediction capability common to all modern high-performance CPUs. The penalty paid (extra clock cycles) for a mispredicted branch can be used for cryptanalysis of cryptographic primitives that employ a data-dependent program flow. Analogous to the recently described cache-based side-channel attacks our attacks also allow an unprivileged process to attack other processes running in parallel on the same processor, despite sophisticated partitioning methods such as memory protection, sandboxing or even virtualization. In this paper, we will discuss several such attacks for the example of RSA, and experimentally show their applicability to real systems, such as OpenSSL and Linux. Moreover, we will also demonstrate the strength of the branch prediction side-channel attack by rendering the obvious countermeasure in this context *(Montgomery Multiplication with dummy-reduction)* as useless. Although the deeper consequences of the latter result make the task of writing an efficient and secure modular exponentiation (or scalar multiplication on an elliptic curve) a challenging task, we will eventually suggest some countermeasures to mitigate branch prediction side-channel attacks.

Keywords: Branch Prediction, Modular Exponentiation, Montgomery Multiplication, RSA, Side Channel Analysis, Simultaneous Multithreading.

1 Introduction

The contradictory requirement of increased clock-speed with decreased power-consumption for today's computer architectures makes branch predictors an

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 225–242, 2007.

inevitable central CPU ingredient, which significantly determines the so called *Performance per Watt* measure of a high-end CPU, cf. [GRAB]. Thus, it is not surprising that there has been a vibrant and very practical research on more and more sophisticated branch prediction mechanisms, cf. [PH, Sha, She].

Unfortunately, the present paper identifies branch prediction even in the presence of recent security promises for commodity platforms from the Trusted Computing area as a novel and unforeseen security risk. Indeed, although even the most recently found security risks for x86-based CPU's have been implicitly pointed out in the old but thorough x86-architecture security analysis, cf. [SPL], we have not been able to find any hint in the literature spotting branch prediction as an obvious side channel attack victim. Let us elaborate a little bit on this connection between side-channel attacks and modern computer-architecture ingredients.

So far, typical targets of side-channel attacks have been mainly Smart Cards, cf. [CNK, Koc]. This is due to the ease of applying such attacks to smart cards. The measurements of side-channel information on smart cards are almost "noiseless", which makes such attacks very practical. On the other side, there are many factors that affect such measurements on real commodity computer systems based upon the most successful one, the Intel x86-architecture, cf. [Sha]. These factors create noise, and therefore it is much more difficult to develop and perform successful attacks on such "real" computers within our daily life. Thus, until very recently the side-channel vulnerability of systems running on servers was not "really" considered to be a danger. This was changed with the work of Brumley and Boneh, cf. [BB], who demonstrated a remote timing attack over a real local network. They simply adapted the attack principle as introduced in [Sch] to show that the RSA implementation of OpenSSL [open] — the most widely used open source crypto library — was not immune to such attacks.

Even more recently, we have seen an increased research effort on the security analysis of the daily life PC platforms from the side-channel point of view. Here, it has been especially shown that the cache architecture of modern CPU's creates a significant security risk, cf. [ASK, Ber, NS, OST06, Per], which comes in different forms. Although the cache itself has been known for a long time being a crucial security risk of modern CPU's, cf. [SPL, Hu], the above papers were the first proving such vulnerabilities practically and raised large public interest in such vulnerabilities.

Especially in the light of the ongoing Trusted Computing efforts, cf. [TCG], which promise to turn the commodity PC platform into a trustworthy platform, cf. also [CEPW, ELMP$^+$, Gra, Pea, TCG, UNRS$^+$], the formerly described side channel attacks against PC platforms are of particular interest. This is due to the fact that side channel attacks have been completely ignored by the Trusted Computing community so far. Even more interesting is the fact that all of the above pure software side channel attacks also allow a totally unprivileged process to attack other processes running in parallel on the same processor (or even remote), despite sophisticated partitioning methods such as memory protection, sandboxing or even virtualization. This particularly means that side channel

attacks render all of the sophisticated protection mechanisms as described in e.g. [Gra, UNRS$^+$] as useless. The simple reason for the failure of these trust mechanisms is that the new side-channel attacks simply exploit deeper processor ingredients — i.e., below the trust architecture boundary, cf. [PL, Gra].

Having said all this, it is natural to identify other modern computer architecture ingredients which have not yet been discovered as a security risk and which are operating below the current trust architecture boundaries. That is the focus of the present paper — a processor's Branch Prediction Unit (BPU). More precisely, we will analyze BPUs and highlight the security vulnerabilities associated with their opaque operations deep inside a processor. In other words, we will present so called branch prediction attacks on simple RSA implementations as a case study to describe the basics of the novel attacks an adversary can use to compromise the security of a platform. Our attacks can also be adapted to other RSA implementations and/or other public-key systems like ECC. We try to refer to specific vulnerable implementations throughout this text.

The paper is organized as follows. We will first give some background information including the structure and functionality of a general BPU and the details of the used RSA-implementations in the next section. Then the following section presents four different attack principles to exploit a BPU in order to break some standard RSA implementations. To do so, we gradually develop from an obvious attack principle more sophisticated attack principles having the potential to break even the implementations that are believed to be immune to side channel attacks. This section is then complemented by presenting the results of some practical implementations of our various attack scenarios. To fully articulate the strength of our attack principles, we have only chosen to implement such attack scenarios which are easy to achieve/expect in practice but having greatest practical significance. Hereafter, we draw some conclusions from the paper and point the reader to further interesting research areas.

2 Background, Definitions and Preliminaries

Although it is beneficial — in order to completely understand our attacks as described later — to know many details about modern computer architecture and branch prediction schemes, it would be unrealistic to explain all these subtle details here. Thus, we refer the reader to [PH, Sha, She] for a thorough treatment of this topics. Nevertheless, we will now explain the basic concepts common to any branch prediction unit, although the exact details differ from processor to processor and are not completely documented in freely available processor manuals.

2.1 Branch Prediction Unit

Superscalar processors have to execute instructions speculatively to overcome control hazards, cf. [She]. The negative effect of control hazards on the effective machine performance increases as the depth of pipelines increases. This fact

makes the efficiency of speculation one of the key issues in modern superscalar processor design. The solution to improve the efficiency is to speculate on the most likely execution path. The success of this approach depends on the accuracy of branch prediction. Better branch prediction techniques improve the overall performance a processor can achieve, cf. [She].

A *branch instruction* is a point in the instruction stream of a program where the next instruction is not necessarily the next sequential one. There are two types of branch instructions: unconditional branches (e.g. jump instructions, goto statements, etc.) and conditional branches (e.g. if-then-else clauses, for and while loops, etc.). For conditional branches, the decision to take or not to take the branch depends on some condition that must be evaluated in order to make the correct decision. During this evaluation period, the processor speculatively executes instructions from one of the possible execution paths instead of stalling and awaiting for the decision to come through. Thus, it is very beneficial if the branch prediction algorithm tries to predict the most likely execution path in a branch. If the prediction is true, the execution continues without any delays. If it is wrong, which is called a *misprediction*, the instructions on the pipeline that were speculatively issued have to be dumped and the execution starts over from the mispredicted path. Therefore, the execution time suffers from a misprediction. The misprediction penalty obviously increases in terms of clock cycles as the depth of pipeline extends. To execute the instructions speculatively after a branch, the CPU needs the following information:

- *The outcome of the branch.* The CPU has to know the outcome of a branch, i.e., taken or not taken, in order to execute the correct instruction sequence. However, this information is not available immediately when a branch is issued. The CPU needs to execute the branch to obtain the necessary information, which is computed in later stages of the pipeline. Instead of awaiting the *actual* outcome of the branch, the CPU tries to predict the instruction sequence to be executed next. This prediction is based on the history of the same branch as well as the history of other branches executed just before the current branch, cf. [She].
- *The target address of the branch.* The CPU tries to determine if a branch needs to be taken or not taken. If the prediction turns out to be taken, the instructions in the target address have to be fetched and issued. This action of fetching the instructions from the target address requires the knowledge of this address. Similar to the outcome of the branch, the target address may not be immediately available too. Therefore, the CPU tries to keep records of the target addresses of previously executed branches in a buffer, the so called *Branch Target Buffer (BTB).*

Overall common to all *Branch Prediction Units (BPU)* is the following Figure 1. As shown, the BPU consists of mainly two "logical" parts, the BTB and the predictor. As said already above, the BTB is the buffer where the CPU stores the target addresses of the previously executed branches. Since this buffer is limited in size, the CPU can store only a number of such target addresses, and

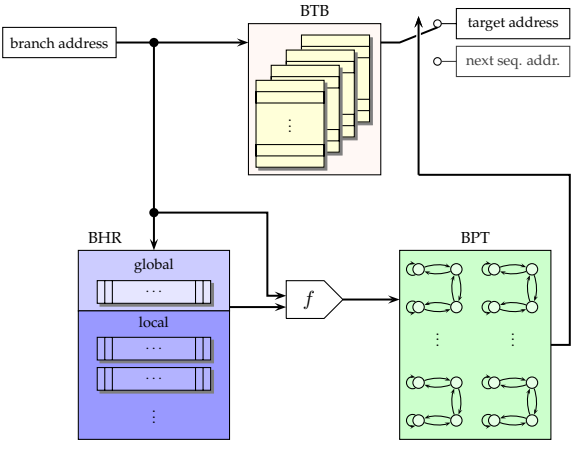

Fig. 1. Branch Prediction Unit Architecture

previously stored addresses may be evicted from the buffer if a new address needs to be stored instead.

The predictor is that part of the BPU that makes the prediction on the outcome of the branch under question. There are different parts of a predictor, i.e., Branch History Registers (BHR) like the global history register or local history registers, and branch prediction tables, cf. [She].

2.2 Details of Popular RSA Implementations

RSA is the most widely used public key cryptosystem which was developed by Rivest, Shamir and Adleman, cf. [MvOV]. The main computation in RSA decryption/signing is the modular exponentiation $P = M^d(\mathrm{mod}N)$, where M is the message or ciphertext, d is the private key that is a secret, and N is the public modulus. Here, N is a product of two large primes p and q. If an adversary obtains the secret value d, he can read all encrypted messages and impersonate the owner of the key. Therefore, the usual main purpose of using timing attacks is to reveal this secret value. If the attacker can factorize N, i.e., he can obtain either p or q, the value of d can be easily calculated. Hence, the attacker tries to find p, q, or d. Since the size of the key is very large, e.g., 1024 bits, the exponentiation is very expensive in terms of the execution time. Therefore, actual implementations of RSA employ efficient algorithms to calculate the result of this operation. In the next subsections we explain the most widely used algorithms.

Binary Square-and-Multiply Exponentiation Algorithm. The binary version of Square-and-Multiply Algorithm (SM) is the simplest way to perform an exponentiation. We want to compute $M^d(mod\ N)$, where d is an n-bit number, i.e., $d = (d_0, d_1, ..., d_{n-1})_2$. Figure 2 shows the steps of SM, which processes

```
S = M
for   i from 1 to n − 1 do
      S = S ∗ S (mod N)
      if   d_i = 1 then
           S = S ∗ M (mod N)
return S
```

Fig. 2. Binary version of Square-and-Multiply Exponentiation Algorithm

the bits of d from left to right. The reader should note that all of the multiplications and squarings are shown as modular operations, although basic SM algorithm computes regular exponentiations. This is because RSA performs modular exponentiation, and our focus here is only on RSA. In an efficient RSA implementation all of the multiplications and squarings are actually performed using a special modular multiplication algorithm, so called Montgomery Multiplication, which we will also recall now.

Montgomery Multiplication. Montgomery Multiplication (MM) is the most efficient algorithm to compute a modular multiplication [MvOV]. It simply uses additions and divisions by powers of 2, which can be accomplished by shifting the operand to the right. Since it eliminates time consuming integer divisions, the efficiency of the algorithm is superior to straightforward school-book methods. Montgomery Multiplication is used to calculate $Z = A * B * R^{-1} \pmod N$, where A and B are the "N-residues" of a and b with respect to R, R is a constant power of 2, and R^{-1} is the inverse of R in modulus N. I.e., $A = a * R \pmod N$, $B = b * R \pmod N$, $R^{-1} * R = 1 \pmod N$, and $R > N$. Another constraint is that R has to be relatively prime to N. But since N is a product of two large primes in RSA, choosing an R of a power of 2 is sufficient to guarantee that these two numbers are relatively prime. Let N be a k-bit odd number, then 2^k is the most suitable value for R. A conversion to and from N-residue format is required to use this algorithm. Hence, it is more attractive to be used for repeated multiplications on the same residue, just like modular exponentiations. Figure 3 shows the steps of Montgomery Multiplication Algorithm. The conditional subtraction $S - N$ on the third line is called "extra reduction" and is of particular interest for Side Channel Attacks in general, cf. [DKLMQ, Sch].

```
S = A ∗ B
S = (S − (S ∗ N^{-1} mod R) ∗ N)/R
if S > N then S = S − N
return S
```

Fig. 3. Montgomery Multiplication Algorithm

3 Outlines of Various Attack Principles

We will gradually develop 4 different attack principles in this section. Although we describe these attacks on a simple RSA implementation, the underlying ideas can be used to develop similar attacks on different implementations of RSA and/or on other ciphers based upon ECC. In order to do so we will assume that an adversary knows every detail of the BPU architecture as well as the implementation details of the cipher (Kerckhoffs' Principle). This is indeed a valid assumption as the BPU details can be extracted using some simple benchmarks like the ones given in [MMK].

3.1 Attack 1 — Exploiting the Predictor Directly (Direct Timing Attack)

In this attack, we rely on the fact that the prediction algorithms are deterministic, i.e., the prediction algorithms are predictable. We present a simple attack below, which demonstrates the basic idea behind this attack. The presented attack is a modified version of Dhem et al.'s attack [DKLMQ]. Assume that the RSA implementation employs Square-and-Multiply exponentiation and Montgomery Multiplication. Assume also that an adversary knows the first i bits of d and is trying to reveal d_i. For any message m, he can simulate the first i steps of the operation and obtain the intermediate result that will be the input of the $(i+1)^{\text{th}}$ squaring. Then, the attacker creates 4 different sets M_1, M_2, M_3, and M_4, where

$M_1 = \{m \mid m$ causes a misprediction during MM of $(i+1)^{\text{th}}$ squaring if $d_i = 1\}$
$M_2 = \{m \mid m$ does not cause a misprediction during MM of $(i+1)^{\text{th}}$ squaring if $d_i = 1\}$
$M_3 = \{m \mid m$ causes a misprediction during MM of $(i+1)^{\text{th}}$ squaring if $d_i = 0\}$
$M_4 = \{m \mid m$ does not cause a misprediction during MM of $(i+1)^{\text{th}}$ squaring if $d_i = 0\}$,

and MM means Montgomery Multiplication. If the difference between timing characteristics, e.g., average execution time, of M_1 and M_2 is more significant than that of M_3 and M_4, then he guesses that $d_i = 1$. Otherwise d_i is guessed to be 0. To express the above idea more mathematically, we define:

- An Assumption A_t^i: $d_i = t$, where $t \in \{0, 1\}$.
- A Predicate $\mathbf{P} : (m) \rightarrow \{0, 1\}$ with

$$\mathbf{P}(m) = \begin{cases} 1 & \text{if a misprediction occurs during the computation of } m^2 (\mathrm{mod} N) \\ 0 & \text{otherwise.} \end{cases}$$

- An Oracle $\mathbf{O}_t : (m, i) \rightarrow \{0, 1\}$ under the assumption A_t^i:

$$\mathbf{O}_t(m, i) = \begin{cases} 1 & \mathbf{P}(m_{temp}) = 1 \\ 0 & \mathbf{P}(m_{temp}) = 0, \end{cases}$$

  where $m_{temp} = m^{(d_0, d_1, \ldots d_{i-1}, t)_2} (\mathrm{mod} N)$.
- A Separation $\mathbf{S}_t^i$ under the assumption A_t^i:

$$(S_0, S_1) = (\{m | \mathbf{O}_t(m, i) = 0\}, \{m | \mathbf{O}_t(m, i) = 1\}).$$

For each bit of d, starting from d_1, the adversary performs two partitionings based on the assumptions A_0^i and A_1^i, where d_i is the next unknown bit that he wants to predict. He partitions the entire sample into two different sets. Each assumption and each plaintext, M, in one of these sets yields the same result for $\mathbf{O}_t(M, i)$. We call these partitioning separations $\mathbf{S}_0^i$ and $\mathbf{S}_1^i$. Depending on the actual value of d_i, one of the assumptions A_0^i and A_1^i will be correct. We define the separation under the correct assumption as "Correct Separation" and the other as "Random Separation". I.e., we define the Correct Separation CS_i as

$$\mathbf{CS}_i = \mathbf{S}_t^i = (CS_0^i, CS_1^i) = (\{M|\mathbf{O}_t(M, i) = 0\}, \{M|\mathbf{O}_t(M, i) = 1\}),$$

and the Random Separation RS_i as

$$\mathbf{RS}_i = \mathbf{S}_{1-t}^i = (RS_0^i, RS_1^i) = (\{M|\mathbf{O}_{1-t}(M, i) = 0\}, \{M|\mathbf{O}_{1-t}(M, i) = 1\}),$$

where $d_i = t$. The decryption of each plaintext in CS_1^i encounters a misprediction delay during the i^{th} squaring, whereas none of the plaintext in CS_0^i results in a misprediction during the same computation. Therefore, the adversary will realize a significant timing difference between these two sets and he can predict the value of d_i. On the other hand, the occurrences of the mispredictions will be random-like for the sets RS_0^i and RS_1^i, which is the reason why we call it a random separation. We can define a correct decision as taking that decision $d_i = t$, where $\mathbf{O}_t(M, i) = \mathbf{P}(M^{(d_0,d_1,\ldots d_i)_2} (\mathrm{mod} N), i)$ for each possible M.

This attack requires the knowledge of the BPU state just before the decryption, since this state, as well as the execution of the cipher, determines the prediction of the target branch. This information is not readily available to an adversary. However, he can perform the analysis phase assuming each possible state one at a time. One expects that only under the assumption of the correct state the above separations should yield a significant difference. Yet, a better approach is to set the BPU state manually. If the adversary has access to the machine the cipher is running on, he can execute a process to reset the BPU state or to set it to a desired state. This will be the strategy for our other attacks. This type of attacks can be applied on any platform as long as a deterministic branch prediction algorithm is used on it. To break a cipher using this kind of attack, we need to have a target branch, outcome of which must depend on the secret/private key of the cipher, a known nonconstant value like the plaintext or the ciphertext, and (possibly) some unknown values that can be searched exhaustively in a reasonable amount of time.

Examples of vulnerable systems. RSA with MM and without CRT (Chinese Remainder Theorem) are susceptible to this kind of attack. The conditional branch of the extra reduction step can be used as the target branch. We have already showed the attack on S&M exponentiation. It can be adapted to b-ary and sliding windows exponentiation, cf. [MvOV], too. In these cases, the adversary needs to search each window value exhaustively and construct the partitions for each of these candidate window values. He encounters the correct separation only for the correct candidate and therefore can realize the correct value of the

windows. If CRT is employed in the RSA implementation, we cannot apply this attack. The reason is that the outcome of the target branch will also depend on the values of p and q, which are not feasible to be searched exhaustively. Similarly, if the RSA implementation does not have a branch that is taken or not taken depending on a known nonconstant value (e.g. extra reduction step in Montgomery Multiplication, which is input dependent to be performed), we cannot use this approach to find the secret key. For example, the if statement in S&M exponentiation (c.f. Line 4 in Fig. 2) as our target branch is not vulnerable to this attack. This is due to the fact that the mispredictions will occur in exactly the same steps of the exponentiation regardless of the input values, and one set in each of the two separations will always be empty.

3.2 Attack 2 — Forcing the BPU to the Same Prediction (Asynchronous Attack)

In this attack we assume that the cipher runs on a simultaneous multi-threading (SMT) machine, cf. [She], and the adversary can run a dummy process simultaneously with the cipher process. In such a case, he can clear the BTB via the operations of the dummy process and causes a BTB miss during the execution of the target branch. The BPU automatically predicts the branch as not to be taken if it misses the target address in the BTB. Therefore, there will be a misprediction whenever the actual outcome of the target branch is 'taken'. We stress that the two parallel threads are isolated and share only the common BPU resource, cf. [She, Sha, OST06, Per]. Borrowed from [OST06] we named this kind of attack an Asynchronous Attack, as the adversary-process needs *no* synchronization with the simultaneous crypto process. Here, an adversary also does not need to know any detail of the prediction algorithm. He can simulate the exponentiations as done in the previous attack and can partition the sample based on the "actual" outcome of the branch. In other words, the following predicate in the oracle (c.f. Section 3.1) can be used:

$$\mathbf{P}(m) = \begin{cases} 1 & \text{if the target branch is taken during the computation of } m^2 (\mathrm{mod} N) \\ 0 & \text{otherwise.} \end{cases}$$

The adversary does not have to clear the entire BTB, but only that BTB set that stores the target address of the branch under consideration, i.e., the target branch. We define three different ways to achieve this:

- *Total Eviction Method:* the adversary clears the entire BTB continuously.
- *Partial Eviction Method:* the adversary clears only a part of the BTB continuously. The BTB set that stores the target address of the target branch has to be in this part.
- *Single Eviction Method:* the adversary continuously clears only the single BTB set that stores the target address of the target branch.

The easiest method to apply is clearly the first one, because an adversary does not have to know the specific address of the target branch. Recall that the

set of the BTB to store the target address of a branch is determined by the actual logical address of that branch. The resolution of clearing the BTB plays a crucial role in the performance of the attack. We have assumed so far that it was possible to clear the entire BTB between two consecutive squaring operations of an exponentiation. However, in practice this is not (always) the case. Clearing the entire BTB may take more time than it takes to perform the operations between two consecutive squarings. Although, this does not nullify the attack, it will mandate (most likely) a larger sample size. Therefore, if an adversary can apply one of the last two eviction methods, he can improve the performance of the attack. We want to mention that, from the cryptographic point of view, we can assume that an adversary knows the actual address of any branch in the implementation due to Kerckhoff's Principle. Under this assumption, the adversary can apply the single eviction method and achieve a very low resolution, which enables him to cause a BTB miss each time the target branch is executed. Recall also, that there is no complicated synchronization between crypto and adversary process needed.

Examples of vulnerable systems. The same systems vulnerable to the first attack (c.f. Section 3.1) are also vulnerable to this kind of attack. The main difference of this attack compared to the first one is the ease of applying it, i.e., unnecessity of knowing = reverse-engineering the subtle BPU details, yielding the correct BPU states for specific time points.

3.3 Attack 3 — Forcing the BPU to the Same Prediction (Synchronous Attack)

In the previous attack, we have specifically excluded the synchronization issue. However, if the adversary finds a way to establish a synchronization with the cipher process, i.e., he can determine for (e.g.) the i^{th} step of the exponentiation and can clear the BTB just before the i^{th} step, then he can introduce misprediction delays at certain points during the computation. Borrowed again from [OST06], we named this kind of attack a Synchronous Attack, as the adversary-process needs some kind of *synchronization* with the simultaneous crypto process. Assume that the RSA implementation employs S&M exponentiation and the if statement in S&M exponentiation (c.f. Line 4 in Fig. 2) is used as the target branch. As stated above, the previous attacks cannot break this system if only the mentioned conditional branch is examined. However, if the adversary can clear the BTB set of the target branch (c.f. Single Eviction Method in Section 3.2) just before the i^{th} step, he can directly determine the value of d_i in the following way.

The adversary runs RSA for a *known* plaintext and measures the execution time. Then he runs it again for the same input but this time he clears the single BTB set *during* the decryption just before the i^{th} execution of the conditional branch under examination, i.e., the if statement of Line 4 in Fig. 2. This conditional branch is taken or not taken depending only on the value of d_i. If it turns out to be taken, the second decryption will take longer time than the first

execution because of the misprediction delay. Therefore, the adversary can easily determine the value of this bit by successively analyzing the execution time.

Examples of vulnerable systems. Any implementation of a cryptosystem is vulnerable to this kind of attack if the execution flow is "key-dependent". The exponents of RSA with S&M exponentiation can be directly obtained even if the CRT is used. If RSA employs sliding window exponentiation, then we can find a significant number of bits (but not all) of the exponents. However, if b-ary method is employed, then only 1 over 2^{wsize} of the exponent bits can be discovered, where $wsize$ is the size of the window. This attack can even break such prominent and efficient implementations that had been considered to be immune to certain kinds of side-channel attacks, cf. [JY, Wal].

3.4 Attack 4 — Trace-Driven Attack Against the BTB (Asynchronous Attack)

In the previous three attacks, we have considered analyzing the execution time of the cipher. In this attack, we will follow a different approach. Again, assume that an adversary can run a spy process simultaneously with the cipher. This spy process continuously executes branches and all of these branches map to the same BTB set with the conditional branch under attack. In other words, there is a conditional branch (under attack) in the cipher, which processes the exponent and executes the corresponding sequence of operations. Moreover, assume also that the branches in the spy process and the cipher process can only be stored in the same BTB set. Recall that it is easy to understand the properties of the BTB using simple benchmarks as explained in [MMK].

The adversary starts the spy process before the cipher, so when the cipher starts decryption/signing, the CPU *cannot* find the target address of the target branch in BTB and the prediction *must* be *not-taken*, cf. [She]. If the branch turns out to be taken, then a misprediction will occur and the target address of the branch needs to be stored in BTB. Then, one of the spy branches has to be evicted from the BTB so that the new target address can be stored in. When the spy-process re-executes its branches, it will encounter a misprediction on the branch that has just been evicted. If the spy-process also measures the execution time of its branches (altogether), then it can detect whenever the cipher modifies the BTB, meaning that the execution time of these spy branches takes a little longer than usual. Thus, the adversary can simply determine the complete execution flow of the cipher process by continuously performing the same operations, i.e., just executing spy branches and measuring the execution time. He will see the prediction/misprediction trace of the target branch, and so he can determine the execution flow. We named this kind of attack an Asynchronous Attack, as the adversary-process needs *no* synchronization at all with the simultaneous crypto process — it is just following the paradigm: continuously execute spy branches and measure their execution time.

Examples of vulnerable systems. Any implementation that is vulnerable to the previous attack is also vulnerable to this one. Specifically any implementation

of a cryptosystem is vulnerable to this kind of attack if the execution flow is "key-dependent". This attack, on the other hand, is very easy to apply, because the adversary does not have to solve the synchronization problem at all. Considering all these aspects of the current attack, we can confidently say that it is a powerful and practical attack, which puts many of the current public-key implementations in danger.

4 Practical Results

We also performed practical experiments to validate the aforementioned attacks which exploit the branch predictor behavior of modern microprocessors. Obviously, eviction-driven attacks using simultaneous multithreading are more general, and demand nearly no knowledge about the underlying BPU — compared to the other type of branch prediction attacks from above. Thus, we have chosen to carry out our experimental attacks in a popular simultaneous multithreading environment, cf. [Sha]. In this setting, an adversary can apply this kind of attacks without any knowledge on the details of the used branch prediction algorithm and BTB structure. Therefore we decided to implement our two asynchronous attacks and show their results as a "proof-of-concept".

4.1 Results for Attack 2 = Forcing the BPU to the Same Prediction (Asynchronous Attack)

In this kind of attack we have chosen, for reasons of simplicity and practical significance, to implement the total eviction method. We used a dummy process that continuously evicts BTB entries by executing branches. This process was simultaneously running with RSA on an SMT platform. It executed a large number of branches and evicted each single BTB entry one at a time. This method requires almost no information on the BTB structure. We performed this attack on a very simple RSA implementation that employed square-and-multiply exponentiation and Montgomery multiplication *with* dummy reduction. We used the RSA implementation in OpenSSL version 0.9.7e as a template and made some modifications to convert this implementation into the simple one as stated above. To be more precise, we changed the window size from 5 to 1, turned blinding off, removed the CRT mode, and added the dummy reduction step. The experiments were run under the configuration shown in Table 1. We used random plaintexts generated by the `rand()` and `srand()` functions available in the standard C library. The current time was fed into `srand()` function as the pseudorandom number generation seed. We measured the execution time in terms of clock cycles using the cycle counter instruction `RDTSC`, which is available in user-level.

We generated 10 million random single-block messages and measured their decryption times under a fixed 512-bit randomly generated key. In our analysis phase, we eliminated the outliers and used only 9 million of these measurements. We then processed each of these plaintext and divided them into the

Table 1. The configuration used in the experiments

Operating System:	RedHat workstation 3
Compiler:	gcc version 3.2.3
Cryptographic Library:	OpenSSL 0.9.7e

sets as explained in Section 3.1 and Section 3.2 based on the assumption of the next unknown bit and the assumed outcome of the target branch. Hereafter we calculated the difference of the average execution time of the corresponding sets for each bit of the key except the first two bits. The mean and the standard deviation of these differences for correct and random separations are given in the following Figure 4. This figure shows also on the right side, the raw timing differences after averaging the 9 million measurements into one single timing difference, where a single dot corresponds to the timing difference of a specific exponent bit, i.e., the x-axis corresponds to the exponent bits from 2 to 511. Using the values in Figure 4, we can calculate the probability of successful

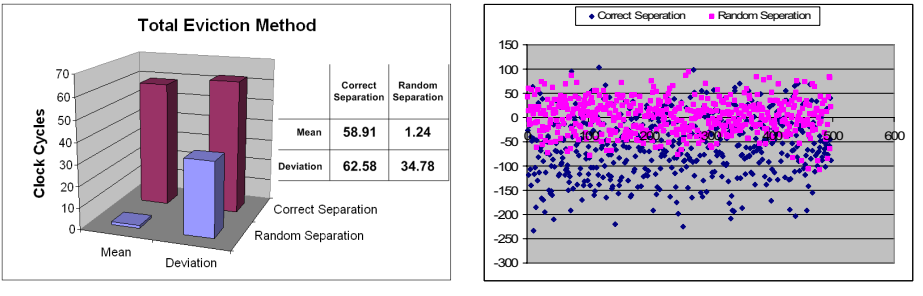

Fig. 4. Practical results when using the total eviction method in attack principle 2

prediction of any single key bit. We interpret the measured average execution time differences for correct and random separation as realizations of normally (Gaussian) distributed random variables, denoted by Y and X respectively. We may assume $Y \sim N(\mu_Y, \sigma_Y^2)$ and $X \sim N(\mu_X, \sigma_X^2)$ for each bit of any possible key, where $\mu_Y = 58.91$, $\mu_X = 1.24$, $\sigma_Y = 62.58$, and $\sigma_X = 34.78$, cf. Figure 4. We then introduce the normally distributed random variable Z as the difference between realizations of X and Y, i.e., $Z = Y - X$ and $Z \sim N(\mu_Z, \sigma_Z^2)$. The mean and deviation of Z can be calculated from those of X and Y as

$$\mu_Z = \mu_Y - \mu_X = 58.91 - 1.24 = 57.67$$
$$\sigma_Z = \sqrt{\sigma_Y^2 + \sigma_X^2} = \sqrt{(62.58)^2 + (34.78)^2} = 71.60$$

As our decision strategy is to pick that assumption of the bit value that yields the highest execution time difference between the sets we constructed under that

assumption, our decision will be correct whenever $Z > 0$. The probability for this realization, $\Pr[Z > 0]$, can be determined by using the z-distribution table, i.e.,

$$\Pr[Z > 0] = \Phi((0 - \mu_Z)/(\sigma_Z)) = \Phi(-0.805) = 0.79,$$

which shows that our decisions will be correct with probability 0.79 if we use $N = 10$ million samples. Although we could increase this accuracy by increasing the sample size, this is not necessary. If we have a wrong decision for a bit, both of the separations will be random-like afterwards and we will only encounter relatively insignificant differences between the separations. Therefore, it is possible to detect an error and recover from a wrong decision without necessarily increasing the sample size.

4.2 Results for Attack 4 = Trace-Driven Attack Against the BTB (Asynchronous Attack)

To practically test attack 4, which is also an asynchronous attack, we used a very similar experimental setup that is described above. But, instead of a dummy process that blindly evicts the BTB entries, we used a real spy function. The spy-process evicted the BTB entries by executing branches just like the dummy process. Additionally, it also measured the execution time of these branches. More precisely, it only evicted the entries in the BTB-set that contains the target address of the RSA branch under attack and reported the timing measurements. In this experiment, we examined the execution of the conditional branch in the exponentiation routine and not the extra reduction steps of Montgomery Multiplication.

We implemented the spy function in such a way that it only checks the BTB at the beginning or early stages of each montgomery multiplication. Thus, we get exactly one timing measurement per montgomery operation, i.e., multiplication or squaring. Therefore, we could achieve a relatively "clean" measurement procedure. We ran our spy and the cipher process N many times, where N is the sample size. Then we averaged the timing results taken from our spy to decrease the noise amplitude in the measurements. The resulting graph shown in Figure 5 presents our first results for different values of N — clearly visualizing the difference between squaring and multiplication.

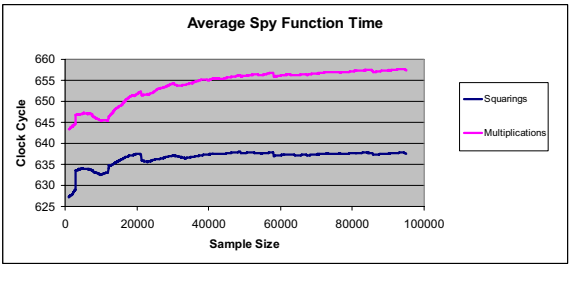

Fig. 5. Increasing gap between multiplication and squaring steps due to missing BTB entries

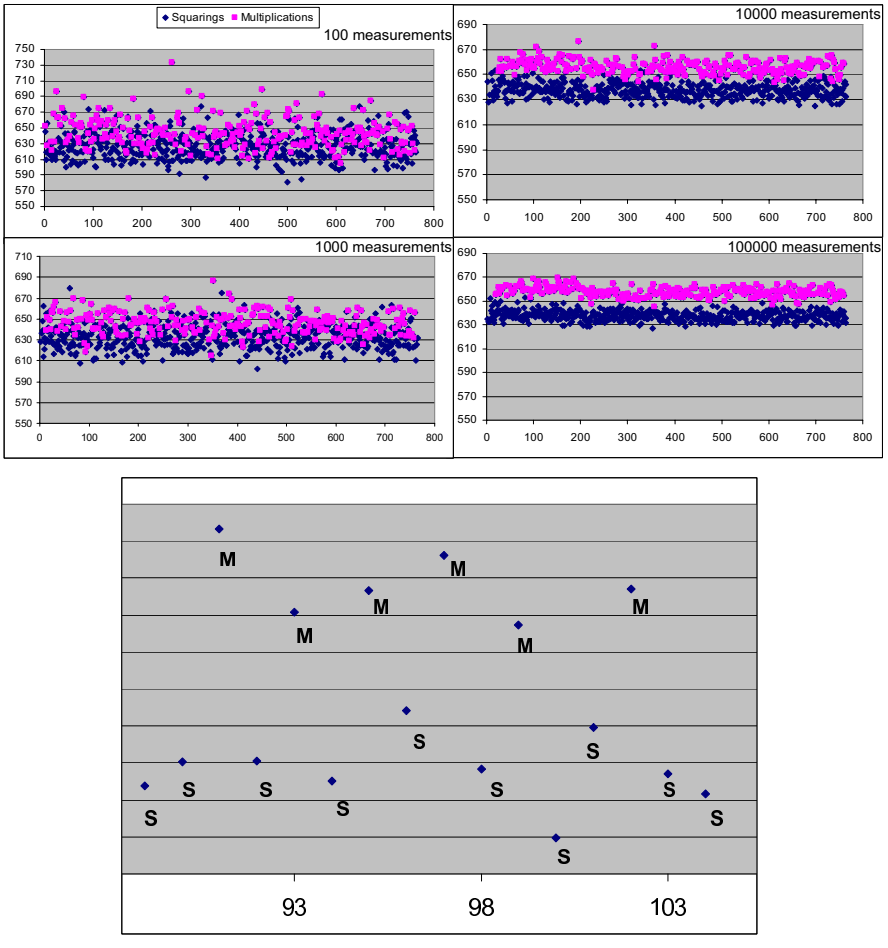

Fig. 6. Connecting the spy-induced BTB misses and the square/multiply cycle gap

As said above, one can deduce very clearly from Figure 5 that there is a stabilizing significant cycle difference between multiplication and squaring steps during the exponentiation. Now, that we have verified this BPU-related gap between the successive multiplication and squaring steps during the exponentiation, we want to show now, how simple it is to retrieve the secret key with this attack principle 4. To do this, we simply zoom into the following Figure 6 with $N = 10000$ measurements. This yields then the picture on the bottom of Figure 6, showing from 89^{th} to 104^{th} montgomery operations for $N = 10000$ measurements. Once such a sequence of multiplications and squarings is captured, it is a trivial task to translate this sequence to the actual values of the key bits.

We would like to remark that the sample size of 10000 measurements might appear quite high at first sight. But using some more sophisticated tricks (which are out of the scope of the present paper) we could have a meaningful square/multiply cycle gap using only a few measurements.

5 Conclusions and Recommendations for Further Research

Along the theme of the recent research efforts to explore software side-channels attacks against commodity PC platforms, this paper has identified the branch prediction capability of modern microprocessors as a new security risk which has not been known so far. Using RSA, the most popular public encryption/signature scheme, and its most popular open source implementation, openSSL, we have shown that there are various attack scenarios how an attacker could exploit a CPU's branch prediction unit. Also, we have successfully implemented a very powerful attack (Attack 4 = Trace-driven Attack against the BTB which even has the power to break prominent side-channel security mechanisms like those proposed by [JY, Wal]). The practical results from our experiments should be encouraging to think about efficient and secure software mitigations for this kind of new side-channel attacks. As an interesting countermeasure the following branch-less exponentiation method, also known as "atomicity" from [CCJ] comes to our mind.

Another interesting research vector might be the idea to apply Branch Prediction Attacks to symmetric ciphers. Although this seems at first sight a bit odd, we would like to point out that an early study of [HK] also applied the Timing Attack scenario of Kocher [Koc] to certain DES implementations and identified branches in the respective DES implementations as a potential source of information leakage. Paired with our improved understanding of branches and their potential information leakage of secrets, it might be a valid idea, to try Branch Prediction Attacks along the ideas of [HK].

Similar to other very recent software side-channel attacks against RSA and AES, cf. [Per, NS, OST06, ASK], our practically simplest attacks rely on a CPU's Simultaneous Multi Threading (SMT) capability, cf. [Sha]. While SMT seems at first sight a necessary requirement of our asynchronous attacks, we strongly believe that this is just a matter of clever and deeper system's programming capabilities and that this requirement could be removed along some ideas as mentioned in [Hu, OST06]. Thus, we think it is of highest importance to repeat our asynchronous branch prediction attacks also on non-SMT capable CPU's.

References

[ASK] O. Acıiçmez, W. Schindler, and Ç. K. Koç. Cache Based Remote Timing Attack on the AES. Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, to appear.

[Ber] D. J. Bernstein. Cache-timing attacks on AES. Technical Report, 37 pages, April 2005. Available at: `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`

[BB] D. Brumley and D. Boneh. Remote Timing Attacks are Practical. *Proceedings of the* 12^{th} *Usenix Security Symposium*, pages 1-14, 2003.

[CEPW] Y. Chen, P. England, M. Peinado, and B. Willman. High Assurance Computing on Open Hardware Architectures. Technical Report, MSR-TR-2003-20, 17 pages, Microsoft Corporation, March 2003. Available at: `ftp://ftp.research.microsoft.com/pub/tr/tr-2003-20.ps`

[CCJ] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. *IEEE Transactions on Computers*, volume 53, issue 6, pages 760-768, June 2004.

[CNK] J.-S. Coron, D. Naccache, and P. Kocher. Statistics and Secret Leakage. *ACM Transactions on Embedded Computing Systems*, volume 3, issue 3, pages 492-508, August 2004.

[DKLMQ] J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestre, J.-J. Quisquater, and J. L. Willems. A Practical Implementation of the Timing Attack. *Proceedings of the 3rd Working Conference on Smart Card Research and Advanced Applications - CARDIS 1998*, J.-J. Quisquater and B. Schneier, editors, Springer-Verlag, LNCS vol. 1820, January 1998.

[ELMP$^+$] P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman. A Trusted Open Platform. *IEEE Computer*, volume 36, issue 7, pages 55-62, July 2003.

[GRAB] S. Gochman, R. Ronen, I. Anati, A. Berkovits, T. Kurts, A. Naveh, A. Saeed, Z. Sperber, and R. Valentine. The Intel Pentium M Processor: Microarchitecture and performance. *Intel Technology Journal*, volume 7, issue 2, May 2003.

[Gra] D. Grawrock. *The Intel Safer Computing Initiative: Building Blocks for Trusted Computing*, Intel Press, 2006.

[HK] A. Hevia and M. Kiwi. Strength of Two Data Encryption Standard Implementations under Timing Attacks. *ACM Transactions on Information and System Security* 2(4):416-437, 1999.

[Hu] W. M. Hu. Lattice scheduling and covert channels. *Proceedings of IEEE Symposium on Security and Privacy*, IEEE Press, pages 52-61, 1992.

[JY] M. Joye and S.-M. Yen. The Montgomery powering ladder. *Cryptographic Hardware and Embedded Systems - CHES 2002*, B. S. Kaliski Jr, Ç. K. Koç, and C. Paar, editors, pages 291-302, Springer-Verlag, LNCS vol. 2523, 2003.

[Koc] P. C. Kocher. Timing Attacks on Implementations of Diffie–Hellman, RSA, DSS, and Other Systems. *Advances in Cryptology - CRYPTO '96*, N. Koblitz, editors, pages 104-113, Springer-Verlag, LNCS vol. 1109, 1996.

[MvOV] A. J. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*, CRC Press, New York, 1997.

[MMK] M. Milenkovic, A. Milenkovic, and J. Kulick. Microbenchmarks for Determining Branch Predictor Organization. *Software Practice & Experience*, volume 34, issue 5, pages 465-487, April 2004.

[open] Openssl: the open-source toolkit for ssl/tls. Available online at: `http://www.openssl.org/`.

[NS] M. Neve and J.-P. Seifert. Advances on Access-driven Cache Attacks on AES. Proceedings of Selected Area of Cryptology (SAC 2006), Montreal, Canada, August 2006, appear at Springer LNCS.

[OST05] D. A. Osvik, A. Shamir, and E. Tromer. Other People's Cache: Hyper Attacks on HyperThreaded Processors. Presentation available at: `http://www.wisdom.weizmann.ac.il/~tromer/`.

[OST06] D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006*, D. Pointcheval, editor, pages 1-20, Springer-Verlag, LNCS vol. 3860, 2006.

[PH] D. Patterson and J. Hennessy. *Computer Architecture: A Quantitative Approach*. 3rd edition, Morgan Kaufmann, 2005.

[Pea] S. Pearson. *Trusted Computing Platforms: TCPA Technology in Context*, Prentice Hall PTR, 2002.

[Per] C. Percival. Cache missing for fun and profit. *BSDCan 2005*, Ottawa, 2005. Available at: `http://www.daemonology.net/hyperthreading-considered-harmful/`.

[PL] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing*, 3rd edition, Prentice Hall PTR, 2002.

[Sch] W. Schindler. A Timing Attack against RSA with the Chinese Remainder Theorem. *Cryptographic Hardware and Embedded Systems - CHES 2000*, Ç. K. Koç and C. Paar, editors, pages 109-124, Springer-Verlag, LNCS vol. 1965, 2000.

[Sha] T. Shanley. *The Unabridged Pentium 4 : IA32 Processor Genealogy*. Addison-Wesley Professional, 2004.

[She] J. Shen and M. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill, 2005.

[SPL] O. Sibert, P. A. Porras, and R. Lindell. The Intel 80x86 Processor Architecture: Pitfalls for Secure Systems. *IEEE Symposium on Security and Privacy*, pages 211-223, 1995.

[Smi1] S. W. Smith. *Trusted Computing Platforms: Design and Applications*, Springer-Verlag, 2004.

[TCG] Trusted Computing Group, `http://www.trustedcomputinggroup.org`.

[UNRS+] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, L. Smith. Intel Virtualization Technology, *IEEE Computer*, volume 38, number 5, pages 48-56, May 2005.

[Wal] C. D. Walter. Montgomery Exponentiation Needs No Final Subtractions. *IEE Electronics Letters*, volume 35, number 21, pages 1831-1832, October 1999.

Template Attacks on Masking—Resistance Is Futile*

Elisabeth Oswald[1,2] and Stefan Mangard[1]

[1] Graz University of Technology, Institute for Applied Information Processing and Communications (IAIK), Inffeldgasse 16a, A–8010 Graz, Austria
{elisabeth.oswald,stefan.mangard}@iaik.tugraz.at
[2] University of Bristol, Department of Computer Science, Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, UK
eoswald@cs.bris.ac.uk

Abstract. In this article we discuss different types of template attacks on masked implementations. All template attacks that we describe are applied in practice to a masked AES software implementation on an 8-bit microcontroller. They all break this implementation. However, they all require quite a different number of traces. It turns out that a template-based DPA attack leads to the best results. In fact, a template-based DPA attack is the most natural way to apply a template attack to a masked implementation. It can recover the key from about 15 traces. All other attacks that we present perform worse. They require between about 30 and 1800 traces. There is no difference between the application of a template-based DPA attack to an unmasked and to a masked implementation. Hence, we conclude that in the scenario of template attacks, masking does not improve the security of an implementation.

1 Introduction

Power analysis attacks and countermeasures are among the most active areas of research in applied cryptography. In this field scientists investigate the practical security of implementations of cryptographic algorithms. In particular, they study how the information that leaks through the power consumption of a cryptographic device can be exploited in practice. It has turned out that unprotected devices, no matter which cryptographic algorithm they implement, are an easy target for power analysis attacks.

In the pioneering work of Kocher *et al.* [KJJ99], simple power analysis (SPA) and differential power analysis (DPA) attacks have been introduced. SPA attacks have been described as attacks in which the attacker has access to only a very small set of power traces of a devices. In contrast, DPA attacks, are based on a large number of power traces. Consequently, the description of DPA attacks includes a statistical method that can be used to determine the unknown key.

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT and through the Austrian Science Fund (FWF) under grant number P18321.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 243–256, 2007.

A particularly interesting class of power analysis attacks, so-called template attacks, has been introduced by Chari *et al.* [CRR03]. In a template-based power analysis attack, the attacker is assumed to know the power consumption characteristics of some instructions of a device. This characterization is stored and called template. Templates are then used as follows. In a template-based DPA attack, the attacker matches the templates based on different key hypotheses with the recorded power traces. The templates that match best indicate the key. This type of attack is the best attack in an information theoretic sense, see [CRR03].

A very popular countermeasure to defeat power analysis attacks is called masking. In a masked implementation, all intermediate values that are computed during the execution of a cryptographic algorithm are concealed by a random value, which is called mask. This ensures, if correctly implemented, a pairwise independence between masked values, unmasked values, and masks. As a consequence, (first-order) DPA attacks do not work.

It has been observed previously, that masked implementations can be attacked by second-order DPA attacks and template-based DPA attacks. However, the research in this area is scattered. Several groups have applied different concepts of attacks to different types of implementations making different assumptions. Consequently, by reading the so-far published literature it is hard to evaluate which attack strategy works best. A related question is how many power traces does a template-based DPA attack need to break a masked implementation? And, continuing this line of thought: is there any difference in the security between an unmasked and a masked implementation in the template scenario?

In this article, we aim to answer these questions. We explain several ways to apply template attacks to masked implementations of block ciphers. These theoretical descriptions are followed by practical results of the implementation of the described attacks. This means, we have implemented all attacks and applied them to an implementation of a masked AES on an 8-bit microcontroller. This allows a fair comparison of the attacks on software implementations on a typical smartcard processor. Our research leads to devastating conclusions for the security of masked implementations of block ciphers on smartcards. In the scenario of template attacks, there is actually no difference in the security of masked and unmasked implementations. Template attacks work virtually in the same way and have the same effectiveness. They can recover the key from about 15 power traces.

This article is organized as follows. Sect. 2 reviews template attacks and discusses related work. In Sect. 3 we explain several template attacks to break masked implementations. In Sect. 4 we apply these attacks to a masked AES software implementation on a microcontroller. We conclude our research in Sect. 5.

2 Template Attacks

The beauty of a standard DPA attack, such as described in [KJJ99], is that it can be conducted by an attacker with relatively low effort. This means, in

a standard DPA attack, the attacker is not assumed to have extensive knowledge about the device and the implementation of the cryptographic algorithm on it. However, in practice also more powerful attackers have to be considered. For instance, there are attackers who are able to obtain detailed knowledge about the power consumption characteristics of a device.

This way of thinking directly leads to the concept of template attacks. A template attack works in two phases. In the first phase, the attacker characterizes a device that is similar to the device that she intends to attack later on. In the second phase, the real attack on a particular device takes place.

2.1 Template Building

During the characterization of a device, the attacker usually determines the probability distribution of the power consumption of certain instructions. Thereby the focus is typically on instructions that are likely to occur during the execution of the cryptographic algorithm, such as different types of move (MOV) instructions or the exclusive-or (XOR) instruction.

In this context it is common to assume that the probability distribution of the power consumption is a multivariate normal distribution. The probability density function of that describes a multivariate normal distribution given in (1).

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2 \cdot \pi)^n \cdot det(\mathbf{C})}} \cdot exp\left(-\frac{1}{2} \cdot (\mathbf{x} - \mathbf{m}) \cdot \mathbf{C}^{-1} \cdot (\mathbf{x} - \mathbf{m})'\right) \quad (1)$$

The multivariate normal distribution is fully defined by the parameter $\mathbf{m}$, which is the mean vector, and the parameter $\mathbf{C}$, which is the covariance matrix. Of course, these two parameters are unknown and have to be estimated by the attacker. Standard estimation methods can be found in textbooks such as [Kay98]. In other words, during the characterization phase, the attacker needs to acquire a number of power traces for each instruction that should be characterized with different data. Then, for each instruction and each data value, the mean vector $\mathbf{m}$ and the covariance matrix $\mathbf{C}$ are estimated by using the acquired power traces. The pair $(\mathbf{m}, \mathbf{C})$ is called *template* and denoted by $\hbar$ in the remainder of this article. As a result of the characterization phase, the attacker has a number of templates: one template $\hbar_v$ for each instruction using a particular data value v. For example, if the attacker has decided to characterize a MOV instruction of an 8-bit microcontroller, the attacker derives 256 templates: one for each 8-bit value that can be moved.

2.2 Template Matching

During the template matching, the attacker calculates intermediate values $v_{i,j}$ of the algorithm by making hypotheses about the key (actually about a part of it). The intermediate values $v_{i,j}$ that depend on the input data d_i $(1 \leq i \leq D)$ and the key hypothesis k_j $(1 \leq j \leq K)$ are associated with the templates. Hence, each key hypothesis k_j suggests a template $\hbar$ for each input value. The

hypothetical key that corresponds to the key in the device always suggests the template that matches best to the acquired trace $\mathbf{t}_i$. Matching a trace $\mathbf{t}_i$ with a template $\hbar$ means that we use (1) to calculate the probability $p(\mathbf{t}_i|k_j)$.

2.3 Template-Based DPA Attack

We start the description of template-based DPA attacks by first only considering one trace $\mathbf{t}_i$. In this case, the attacker is essentially interested in answering the following question: Given the trace $\mathbf{t}_i$, what is the probability that the key of the device equals k_j? This conditional probability $p(k_j|\mathbf{t}_i)$ can be calculated based on the result of the template matching using *Bayes' theorem*, see for example [Kay98]. Bayes' theorem allows us to calculate the probability $p(k_j|\mathbf{t}_i)$ based on the prior probability $p(k_l)$ and the probability $p(\mathbf{t}_i|k_l)$. This is shown in (2).

$$p(k_j|\mathbf{t}_i) = \frac{p(\mathbf{t}_i|k_j) \cdot p(k_j)}{\sum_{l=1}^{K} (p(\mathbf{t}_i|k_l) \cdot p(k_l))} \tag{2}$$

The prior probabilities are the probabilities for the different keys without considering the trace $\mathbf{t}_i$. Bayes' theorem can hence essentially be viewed as an update function for probabilities. The input of the function are the prior probabilities $p(k_l)$ that do not consider $\mathbf{t}_i$. The output are the posteriori probabilities $p(k_j|\mathbf{t}_i)$ that consider $\mathbf{t}_i$. Note that sums of the priori probabilities and the posteriori probabilities are always 1, *i.e.*$\sum_{l=1}^{K} p(k_l) = \sum_{l=1}^{K} p(k_l|\mathbf{t}_i) = 1$.

Given just one trace $\mathbf{t}_i$, the best guess for the key that is used by the device is the key k_j that leads to the highest probability $p(k_j|\mathbf{t}_i)$. Guessing the key of the device based on this strategy is called maximum-likelihood approach.

Now we extend this approach to multiple traces. We are hence interested in determining the probability $p(k_j|\mathbf{T})$. This is the probability that, given a matrix $\mathbf{T} = (t_{i,j})_{D \times T}$ of power consumption values, the device uses the key k_j. The number of rows of $\mathbf{T}$ is D because there is one trace for each input data.

The extension from $p(k_j|\mathbf{t}_i)$ to $p(k_j|\mathbf{T})$ is not very difficult. Since the power traces are statistically independent, we can multiply the probabilities that correspond to different traces and fill the product into (2), see (3). An alternative way to derive this formula is to apply Bayes' rule iteratively. This means $p(k_j|\mathbf{T}) = p(k_j|\mathbf{t}_D)$, if the prior probability $p(k_j)$ is set to $p(k_j|\mathbf{t}_{i-1})$ when calculating $p(k_j|\mathbf{t}_i)$.

$$p(k_j|\mathbf{T}) = \frac{\left(\prod_{i=1}^{D} p(\mathbf{t}_i|k_j)\right) \cdot p(k_j)}{\sum_{l=1}^{K} \left(\left(\prod_{i=1}^{D} p(\mathbf{t}_i|k_l)\right) \cdot p(k_l)\right)} \tag{3}$$

Equation (3) is the basis for template-based DPA attacks. It leads to the probabilities $p(k_j|\mathbf{T})$ for $j = 1, \ldots, K$ that can be used to guess the key of the device based on the maximum-likelihood approach.

Note, that there are some practical issues that need to be taken into account. For instance, the number of points within a trace, which are used to build the templates, needs to be chosen carefully. One might argue that the more points

the better. However, the size of the covariance matrix grows quadratically in the number of points. In addition, evaluating (1) requires to invert the covariance matrix which often causes numerical problems, especially if the matrix is large. Consequently, one needs to be careful when selecting the points that define the template. In the literature, one commonly refers to those points as *points of interest*. References [CRR03] and [RO04] have discussed ways to find a good set of points of interest. The approach taken by these article is simple: one uses DPA attacks to find the points of interest. A trick that can help to cope with problems associated with the covariance matrix is to work with *reduced templates*. In a reduced template, the covariance matrix is simply the identity matrix.

2.4 Related Work

Template attacks have been originally proposed by Chari *et al.* [CRR03]. They have been applied in an SPA attack to break an RC4 implementation. In a later work by Agrawal *et al.* [ARR03] this concept has been expanded and discussed in the context of DPA attacks. Finally, Agrawal *et al.* [ARRS05] have provided a first example of using templates to attack masking schemes. This article is of particular interest because it actually restricts the template assumption. This means, instead of assuming that the attacker has full control (including the mask generation) over the device during the characterization phase, the authors assume that the attacker has only access to a device with a defective random number generator (RNG). Due to the defective RNG, the attacker can find the points of interest with a DPA even though masking is applied. Consequently, templates can be built. Peeters *et al.* [PSDQ05] have applied template attacks to masked hardware implementations. They succeeded, however, with a relatively high number of traces.

In this article, we investigate how template attacks can be applied to break software implementations of masking schemes. We investigate the number of required traces and the applicability in practice. Furthermore, we compare our attacks to previous attacks on masking schemes. The purpose of all these investigations is to find out whether masking improves the security if template attacks are considered.

3 Template Attacks on Masking Schemes

As explained in the previous section, template attacks have been studied in the literature in different contexts. Lately, they have also been applied to implementations of masking schemes. Unfortunately, previous work is scattered in the sense that attacks under specific assumptions were applied to different implementations on different platforms. Hence, it is difficult to assess the true power of template attacks on masking schemes.

Masking implementations of cryptographic algorithms is very popular, especially in theory, but also in practice. Papers with co-authors from the industry such as [AG03] or [BGK05] show that even the industry investigates masking as a practical countermeasure.

In a masked implementation, each intermediate value v is concealed by a random value m that is called mask: $v_m = v * m$. Hence, the mask is generated inside the cryptographic device and varies from execution to execution. It is not known to the attacker. The operation $*$ is typically defined according to the operations that are used in the cryptographic algorithm. In the case of implementations of block ciphers, the operation $*$ is most often the Boolean exclusive-or function $\oplus$.

Masking provides security against first-order DPA attacks, because each masked intermediate value v_m is pairwise independent of v and m. In a typical implementation, the masks are directly applied to the plaintext or the key. The implementation of the algorithm needs to be slightly changed in order to process the masked intermediate values and in order to keep track of the masks. The resulting is also masked. Hence, the masks need to be removed at the end of the computation in order to get the ciphertext.

In this section, we discuss different ways to apply template attacks to break implementations of masking schemes. First, we discuss the most natural way of applying a template attack to a masked implementation. Second, we discuss how templates can be used together with second-order DPA techniques.

3.1 Template-Based DPA Attack

As sketched in Sect. 2, the attacker builds templates for some intermediate operations that occur in the implementation of the cryptographic algorithm. For instance, the attacker could characterize the MOV instruction, which is often used in implementations of block ciphers. Assuming that we characterize an n-bit processor, the attacker produces 2^n templates. During the attack, the template matching takes place. This means, the attacker computes hypothetical intermediate values based on key hypotheses. Recall from the previous section that because each intermediate value is associated with a template, also each key hypothesis is associated with a template. The obstacle that comes into play in a masked implementation is that the intermediate values are masked, and we do not know the value of the mask m in a certain encryption run. This implies that we have to perform the template matching for all M values that the mask m can take. Consequently, the template matching gives the probabilities $p(\mathbf{t}_i|k_j \wedge m)$ and we have to derive $p(\mathbf{t}_i|k_j)$ by calculating (4).

$$p(\mathbf{t}_i|k_j) = \sum_{m=0}^{M-1} p(\mathbf{t}_i|k_j \wedge m) \cdot p(m) \tag{4}$$

With $p(\mathbf{t}_i|k_j)$ we can calculate (3). Hence, except for the extra calculation of (4), a template-based DPA attack on a masked implementation works in exactly the same manner as a template-based DPA attack on an unmasked implementation.

3.2 Template Attacks Combined with Second-Order Techniques

We have already seen that template attacks apply naturally to masked implementations. They are the best attacks to break unmasked implementations [ARR03]. Because they can be applied in the same way to masked implementations, they can be expected to be the best attacks for masked implementations as well. Nevertheless, it is still interesting to see if and how they can be combined with second-order DPA techniques. The reason is to find out whether a combination could lead to attacks with different assumptions. The combinations that we discuss use templates to extract some information from the (pre-processed) traces, and then perform a DPA attack. Before we discuss some ways to combine template and second-order DPA attacks, we review the working principle of second-order DPA attacks first.

Second-Order DPA Attacks. In a second-order DPA attack, information about two intermediate values, which are related to the same mask, is combined. Then, after this combination, a standard DPA attack is performed. Consequently, a second-order DPA attack on a software implementation consists of two steps. In the first step, the preprocessing step, a function that combines points (pairwise) within the acquired power traces is applied to the traces. In the second step, a standard DPA attack with suitable key hypotheses is applied to the preprocessed data. In the case of masked implementations of block ciphers on smartcards, computing the absolute value of the difference of two points has turned out to be a suitable preprocessing function [OMHT06]. Based on the key hypotheses, the attacker computes the exclusive-or between two unmasked intermediate values and uses this as hypothetical intermediate value in the DPA attack. This works nicely for smartcards that leak the Hamming weight (HW) of the intermediate values. In this case, the two functions $HW(u \oplus v)$ and $|HW(u_m) - HW(v_m)|$ are reasonably correlated. More precisely, the correlation coefficient ρ for the correct key ck at the time tc when the attacked intermediate result(s) are computed, is 0.24 if u and v are 8-bit values.

$$\rho_{ck,ct}(HW(u \oplus v), |HW(u_m) - HW(v_m)|) = 0.24$$

Because a second-order DPA attack requires to pre-process the traces, we can build templates before, during, and after this preprocessing step.

Templates Before Preprocessing. Second-order DPA attacks require more traces than standard DPA attacks because the pre-processed data, *i.e.* the data that corresponds to $|HW(a \oplus m) - HW(b \oplus m)|$, correlates only reasonably well to the hypothetical data $HW(a \oplus b)$. The authors of [JPS05] have theoretically and the authors of [OMHT06] have practically assessed that the correlation coefficient $\rho_{ck,ct}$ for a second-order DPA attack with this particular preprocessing method is about 0.24 for an 8-bit device that leaks the Hamming weight. This is still good enough for an attack but significantly less that the correlation coefficient that can be expected for a standard DPA attack on such devices (which can be close to 1, depending on the hypotheses).

Therefore, we would like to find preprocessing functions $pre()$ that maximize $\rho(HW(u \oplus v), pre(HW(u_m), HW(v_m))$. The function $HW(u \oplus v)$ has a complicated structure. When approximating functions with a complicated structure, theory suggests using trigonometric functions. Our experiments have shown that using higher degree polynomials based on the sine function improves the correlation coefficient dramatically. In addition, instead of using $HW(u \oplus v)$ we can use more complex combination functions $comb()$. For example, it turns out that the correlation between the two following sine-based functions is about 0.83 when considering 8-bit values:

$$\begin{aligned} comb(u,v) = &- 89.95 \cdot \sin\left(HW(u \oplus v)^3\right) - \\ &- 7.82 \cdot \sin\left(HW(u \oplus v)^2\right) + 67.66 \\ pre(HW(u_m), HW(v_m)) = &\sin\left(HW(u_m) - HW(v_m)\right)^2 \end{aligned}$$

$$\rho(comb(u,v), pre(HW(u_m), HW(v_m))) = 0.83$$

There is one significant problem when using these improved preprocessing and combination functions in practice. We have derived them based on noise-free data (*i.e.* the Hamming weights). In practice however, we do not have noise free measurements. Hence, the improved preprocessing will not work immediately.

In the light of this, templates can be useful. The idea is to build templates that allow identifying the Hamming weight of the processed data. If such templates are available, a second-order DPA attack works as follows. The templates are used to deduce $HW(u_m)$ and $HW(v_m)$. Instead of doing the preprocessing on the traces, we do the improved preprocessing on the extracted Hamming weights. Because we now work directly with the Hamming weights, we can use an improved preprocessing function and thereby increase the correlation coefficient.

Templates During Preprocessing. The security of any masked implementation depends on the randomness of the masks. Randomness means that the masks are uniformly distributed. Consequently, if the random number generator which produces the masks is biased, masking is insecure. The goal of an attacker could be to force a bias into the masks. This can be done by active attacks, however, a much simpler method has been pointed out in [Jaf06]. If the attacker is able to discard some subset of the acquired traces, which belong to a certain subset of masks, then the remaining traces and the therein used masks must be biased.

Templates can be naturally used in this context. The attacker needs to build templates that can identify for instance the Hamming weight of some intermediate data. During the attack, the attacker uses these templates to identify the Hamming weight of the processed masks. Only those traces that correspond to a subset of the masks, for instance, the traces in which the Hamming weight of the used mask is smaller than six are selected. This is basically the preprocessing step of the second-order DPA attack. A standard DPA on the selected traces reveals the key.

Templates After Preprocessing. In the last scenario, we use templates after preprocessing. This means, the attacker builds templates for $HW(u \oplus v)$ (the

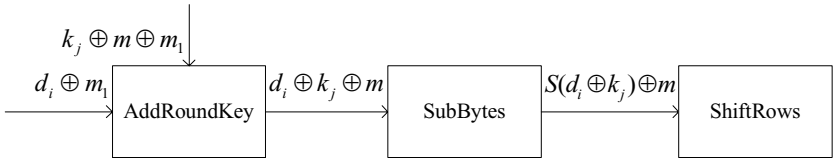

Fig. 1. Part of a masked AES round. The sequence of AddRoundKey, SubBytes and Shiftrows is applied to all bytes of the AES state. We assume that input and output masks of SubBytes are equal.

hypothetical values that are used during the DPA attack) with the preprocessed traces. During the DPA attack, the attacker uses these templates to extract the Hamming weights $HW(u \oplus v)$ and performs a DPA attack on these values.

The preprocessing removes quite an amount of information from the power traces. Recall that the correlation coefficient goes down from 1 to 0.24. Hence, it cannot be expected that this strategy leads to better results than the attacks described before.

4 Results from Attacking a Masked AES Smart Card Implementation

In order to assess the practical value of the attacks that we described in Sect. 3, we have applied them to a masked software implementation of AES on a microcontroller.

In our masked software implementation of AES the inputs and outputs of each operation are masked additively. Like in many software implementations, we first compute AddRoundKey, SubBytes, and ShiftRows for all bytes of the state. Then, MixColums follows. For the sake of simplicity, we will mount all our attacks on the sequence of AddRoundKey, SubBytes and Shiftrows, see Fig.1. We assume that plaintext d_i and key k_j are masked and that the input mask and the output mask of SubBytes are equal. This mask is denoted by m. This is a quite typical implementation of a masked software AES.

The microcontroller that we have used is a standard 8-bit microcontroller which is similar to many microcontrollers found in smartcards. Because it precharges the bus lines, it leaks the Hamming weight of the data. Because of that, it makes no sense to build templates for all 256 data values. Instead, we have built templates for each instruction and all Hamming weights. For this characterization, we have acquired 10000 traces.

In the remainder of this section, we present results from the actual implementation of the attacks that we have described in the previous section.

4.1 Template-Based DPA Attack

In this example, we have built templates that contain the joint leakage of two masked intermediate values. In our AES implementation, several masked

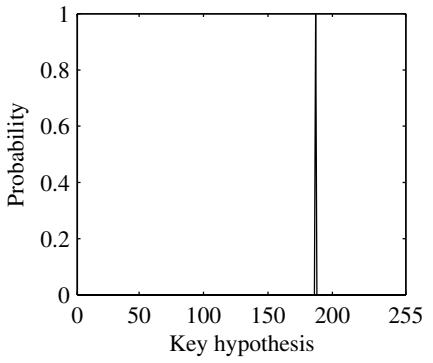

Fig. 2. Result of a template-based DPA attack. The correct key hypothesis has probability one. The incorrect key hypotheses all have probability zero.

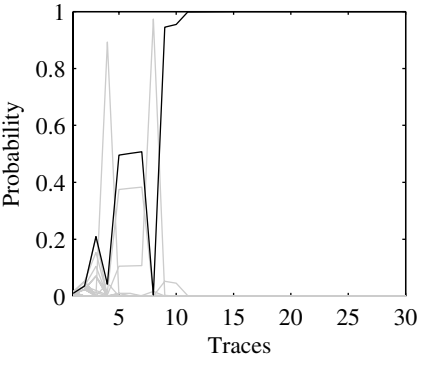

Fig. 3. Evolution of the probability over an increasing number of traces. The correct key hypothesis is plotted in black. The incorrect key hypotheses are plotted in gray.

intermediate values can be used. For instance, the input and output of SubBytes are both masked by m. In addition, at some time at the beginning of the algorithm, the masks have to be generated. Hence, there is another instruction that manipulates m at the beginning of the algorithm. In order to determine the interesting points for our templates, [CRR03] and [RO04] proposed to perform DPA attacks. In this way, we have determined the points in time when m, $d_i \oplus k_j \oplus m$ and $S(d_i \oplus k_j) \oplus m$ are computed.

Our templates take the power model of the microcontroller into account. Hence, we have built 81 templates, one for each pair of $HW(m)$ and $HW(S(d_i \oplus k_j) \oplus m)$:

$$\hbar_{HW(m),HW(S(d_i \oplus k_j) \oplus m)}$$

The template matching then gives the probabilities for $p(\mathbf{t}_i | k_j \wedge m)$:

$$p(\mathbf{t}_i | k_j \wedge m) = p(\mathbf{t}_i; \hbar_{HW(m),HW(S(d_i \oplus k_j) \oplus m)})$$

With these probabilities, and by assuming that $p(m) = 1/M$, we have calculated (4) and subsequently we have derived $p(k_j | \mathbf{T})$ with (3).

The result of this attack is depicted in Figure 2. It shows that one key has probability one whereas all other key hypotheses have probability zero. Figure 3 shows that with about 15 traces the correct key (plotted in black) can be identified. This shows that this template-based DPA attack on a masked AES implementation in software works in the same way as a template attack on an unmasked implementation in software.

4.2 Templates Attacks Combined with Second-Order Techniques

We have also applied the attacks that we have described in Sect. 3.2 to the same implementation using the same set of power traces for characterization.

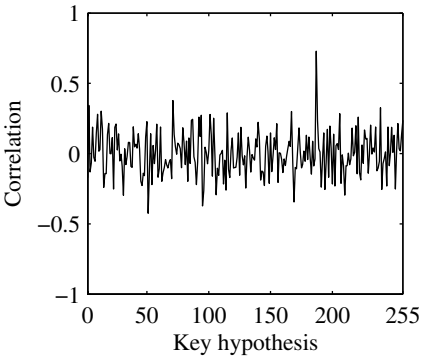

Fig. 4. Result of a second-order DPA attack that uses templates before preprocessing

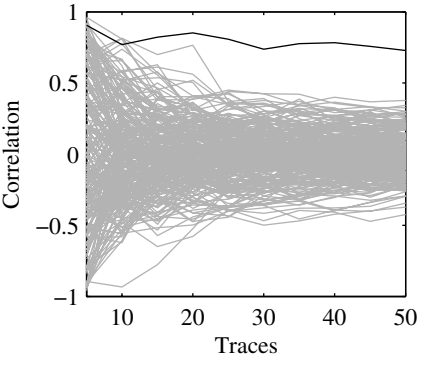

Fig. 5. Evolution of the correlation coefficient over an increasing number of traces

Templates before Preprocessing. In this attack, we use templates to extract the Hamming weights of the masked intermediate values before and after SubBytes. Hence, we obtain $HW((d_i \oplus k_j) \oplus m)$ and $HW(S(d_i \oplus k_j) \oplus m)$ with the templates and apply the improved preprocessing function to these values. The hypotheses are $u = HW(d_i \oplus k_j)$ and $v = S(d_i \oplus k_j)$. Based on these hypotheses, we have calculated the improved combination.

Figure 4 shows the result of a second-order DPA attack using templates before preprocessing. As we have extracted the Hamming weights of the two targeted intermediate values $(d_i \oplus k_j) \oplus m$ and $S(d_i \oplus k_j) \oplus m$ with templates, we only get the correlation coefficient that correspond to them. This is why Figure 4 shows only 256 values (one for each key hypothesis). It is clearly visible that the correct key hypothesis has a correlation coefficient of about 0.83. Figure 5 shows that with about 30 traces, the correct key hypothesis can be distinguished from the incorrect key hypotheses.

Templates during Preprocessing. In this attack, we have used templates to identify the traces in which the mask has a Hamming weight that is smaller than six. In the preprocessing step we have discarded all the traces with this property. Then, we have performed a DPA attack on the remaining traces. Figure 6 shows the result of such an attack. It shows the correlation for all 256 key hypotheses. The line that is plotted in black indicates the correct key hypothesis. Figure 7 shows that with about 450 traces the correct key hypothesis can be identified.

Templates after Preprocessing. In this attack, we have built templates for $HW((d_i \oplus k_j) \oplus S(d_i \oplus k_j))$ based on the preprocessed traces. In the second-order DPA attack we have used these templates to extract $HW((d_i \oplus k_j) \oplus S(d_i \oplus k_j))$. Then, we have mounted a DPA attack on these values. Figure 8 shows the result of such an attack. The figure shows the correlations for all 256 key hypotheses. The correct key hypothesis is indicated by the highest correlation coefficient. Figure 9 shows that about 2 000 traces are necessary to identify the correct key hypothesis. Apparently, this attack requires a large number of power traces.

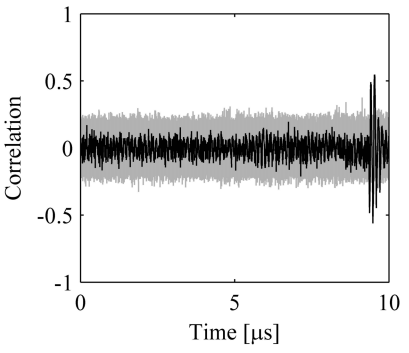

Fig. 6. Result of a second-order DPA attack that uses templates for preprocessing

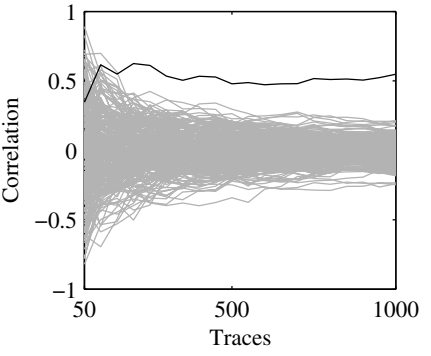

Fig. 7. Evolution of the correlation coefficient over an increasing number of traces

4.3 Comparison of Template Attacks

Our results have confirmed that a template-based DPA attack is the most powerful attack on a masked implementation in practice. We have broken such a masked implementation with about 15 power traces. The other attacks that we have come up with, and the one that has been suggested by [Jaf06], are weaker. These other attacks combine template attacks with second-order DPA techniques.

The first attack that we have described in Sect. 3.2 (the attack using templates to improve the preprocessing in a second-order DPA attack) actually allows us to draw another conclusion about improving preprocessing techniques. More efficient preprocessing techniques require complex functions which are derived based on the idealized power consumption of some intermediate values. Hence in practice, we have to use templates to extract this ideal power consumption from the traces. This means that in practice, improving a second-order DPA attack leads to a template attack. Consequently, instead of doing this attack, we can immediately do a template-based DPA attack.

The second attack that we have described in Sect. 3.2 (the attack where we use templates to bias the masks) is slightly different. It requires to determine only the point in time when the masks are processed. In addition, one can perform this attack with reduced templates, or even templates that extract even less information than the Hamming weight. Remember that we only want to discard a subset of the traces that correspond to some class of masks (low Hamming weight, or high Hamming weight, etc.). Hence, such an attack requires the attacker to have only little knowledge about the device and the implementation of the algorithm. In other words, the attacker does not need an explicit characterization phase.

The last attack that we have discussed in Sect. 3.2 (the attack where we have built templates for the pre-processed traces) works worst as expected. It has the advantage that during characterization, the masks do not need to be known to the attacker. However, the standard second-order DPA attack that we have

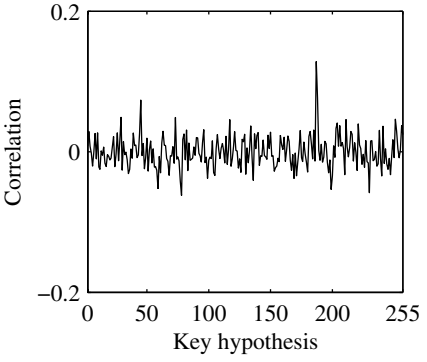

Fig. 8. Result of a second-order DPA attack that uses templates after preprocessing

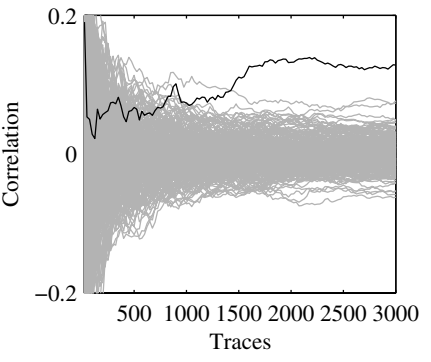

Fig. 9. Evolution of the correlation coefficient over an increasing number of traces

presented in [OMHT06], and that we have performed on the same device using the same measurement setup, required about 450 traces. Hence, this attack does not lead to an improvement over a standard second-order DPA attack.

5 Conclusion

In this article, we have discussed different types of template attacks on masked implementations. All template attacks that we have described could break our masked AES software implementation on an 8-bit microcontroller. However, they all required quite a different amount of traces. It has turned out that a template-based DPA attack leads to the best results. It can recover the key from about 15 traces. All other attacks that we have come up with, and the one by [Jaf06], have performed worse (they required between about 30 and 1800 traces). A template-based DPA attack is the best attack in theory [ARR03] and in practice. Even when applied to a masked implementation, it only requires about 15 traces to recover the key. We conclude that in the scenario of template attacks, masking does not improve the security of an implementation.

References

[AG03] Mehdi-Laurent Akkar and Louis Goubin. A Generic Protection against High-Order Differential Power Analysis. In Thomas Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 192–205. Springer, 2003.

[ARR03] Dakshi Agrawal, Josyula R. Rao, and Pankaj Rohatgi. Multi-channel Attacks. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2003.

[ARRS05] Dakshi Agrawal, Josyula R. Rao, Pankaj Rohatgi, and Kai Schramm. Templates as Master Keys. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2005.

[BGK05] Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably Secure Masking of AES. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2005.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2003.

[Jaf06] Joshua Jaffe. More Differential Power Analysis: Selected DPA Attacks, June 2006. Presented at ECRYPT Summerschool on Cryptographic Hardware, Side Channel and Fault Analysis.

[JPS05] Marc Joye, Pascal Paillier, and Berry Schoenmakers. On Second-Order Differential Power Analysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.

[Kay98] Steven M. Kay. *Fundamentals of Statistical Signal Processing - Detection Theory*. Prentice Hall, 1998. ISBN 0-13-504135-X.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[OMHT06] Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2006.

[PSDQ05] Eric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. Improved Higher-Order Side-Channel Attacks with FPGA Experiments. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 2005.

[RO04] Christian Rechberger and Elisabeth Oswald. Practical Template Attacks. In Chae Hoon Lim and Moti Yung, editors, *Information Security Applications, 5th International Workshop, WISA 2004, Jeju Island, Korea, August 23-25, 2004, Revised Selected Papers*, volume 3325 of *Lecture Notes in Computer Science*, pages 443–457. Springer, 2004.

Differential Power Analysis of Stream Ciphers

W. Fischer, B.M. Gammel, O. Kniffler, and J. Velten

Infineon Technologies AG, Germany

Wieland.Fischer@infineon.com,
Berndt.Gammel@infineon.com,
Oliver.Kniffler@infineon.com,
Joachim.Velten@infineon.com

Abstract. Side-channel attacks on block ciphers and public key algorithms have been discussed extensively. However, there is only sparse literature about side-cannel attacks on stream ciphers. The few existing references mainly treat timing [8] and template attacks [10], or provide a theoretical analysis [6], [7] of weaknesses of stream cipher constructions. In this paper we present attacks on two focus candidates, Trivium and Grain, of the eSTREAM stream cipher project. The attacks exploit the resynchronization phase of ciphers. A novel concept for choosing initial value vectors is introduced, which totally eliminates the algorithmic noise of the device, leaving only the pure side-channel signal. This attack allows to recover the secret key with a small number of samples and without building templates. To prove the concept we apply the attack to hardware implementations of the ciphers. For both stream ciphers we are able to reveal the complete key.

Keywords: side-channel attack, power analysis, DPA, stream cipher, Trivium, Grain.

1 Introduction

Differential power analysis (DPA) is a well-known and thoroughly studied threat for implementations of block ciphers, like DES and AES, and public key algorithms, like RSA. However, in the field of stream ciphers this topic is rather unknown. More generally, this is even true for any kind of side-channel analysis (SCA). Side-channel attacks are built on the fact that cryptographic algorithms are implemented on a physical device. SCA can use all kinds of physical emanation from the device, like current consumption, electromagnetic radiation, or execution time variations. This so-called side-channel may leak information about secret data. Even the regular output of the algorithm can be seen as a side-channel—in the case that an attacker was able to induce a fault into the data or control path of the computation. Although there is vast literature about SCA on implementations of block ciphers and public key algorithms, only few publications can be found about attacks on stream ciphers. In [4] the authors study fault attacks on stream ciphers like LILI-128, RC4, and SOBER-t32. The

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 257–270, 2007.

latter one was the target of a timing attack in [8]. Template attacks, which were introduced in [2], were mounted on RC4 in [9]. So far, there are no reports on a practical DPA targeting a hardware implementation of a stream cipher. There is only one work, [7], which describes theoretically DPA attacks on A5/1 and E0. These are classical DPA attacks which aim at raising the side-channel signal above the algorithmic noise[1] by statistical means (by averaging over many power traces). In contrast, the presented method cancels out the algorithmic noise exactly, by using specially tailored sets of initial value vectors.

Differential power analysis was introduced by Kocher in [5]. In a DPA an attacker generates a set of hypotheses (about some secret value or a partial key) and tries to identify the (unique) true hypothesis by finding the highest correlation between the power consumption of the physical realization of an algorithm and those internal bits which can be computed by the attacker by virtue of one of these hypotheses. The classical setup for a DPA is illustrated in Figure 1. Some parts S of an implementation of a cryptographic algorithm have the characteristic:

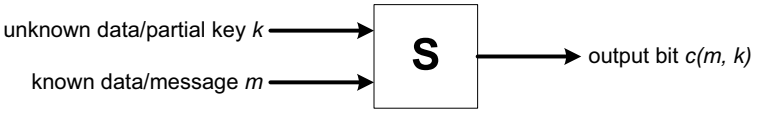

Fig. 1. Generic Setup for DPA

- input: known bits m, a few unknown bits k
- output: a bit $c(m, k)$ with the properties
  - $c(m, k) = c(m, k')$ for all possible m and $k = k'$
  - $c(m, k) = c(m, k')$ for about 50% of all possible values of m if $k \neq k'$—even if k and k' differ only by one bit.

This implies, of course, that S cannot be linear. But in any good cipher one is always able to identify some parts with this property.

Our attack targets the resynchronization phase of stream ciphers. Unlike in [7], were a *known IV DPA attack* is described, we will describe and execute a *chosen IV DPA attack*. It will be shown that the signal-to-noise ratio of the side-channel signal, which carries information about the secret key, can be optimized by specially chosen initial value vectors. These are constructed in such a way, that in the statistical analysis of the power traces contributions to the power consumption, which are not related to the correlation signal, will cancel out. We will elaborate the attacks for two recently published stream ciphers, Grain [3] and Trivium [1]. In the case of the first cipher, we will make partial use of a nonlinear element S. In the case of the second cipher we will *not* use

[1] Algorithmic noise is generated by the power consumption caused by the execution of the algorithm. Contrary to thermal noise it cannot be eliminated by averaging over power traces with identical input parameters.

any nonlinearity. S will rather be an XOR gate. By virtue of the new selection scheme for the initial value vectors we are able to mount practical attacks on hardware implementations of the two ciphers. The attack is efficient in practice, as there is no need to construct templates. Also the number of samples is small.

Outline—We will describe the attacks on both stream ciphers. In each case we will first give a definition of the cipher and shortly describe a straight forward hardware implementation, in order to state a theoretical power model for this implementation. We will describe the actual attack on the cipher and show why the attack works in our chosen power model. Finally the attack on a physical implementation of Grain on a field programmable gate array device (FPGA) will be reported.

2 Differential Power Analysis of Grain

The target of the attack will be the second version of Grain [3]. After a description of the structure and the implementation of the cipher the power model for a CMOS implementation will be defined and the theory for the attack will be elaborated. Finally we will report the results of a practical realization of the attack on a hardware implementation.

2.1 Definition of Grain

Grain is a binary additive synchronous stream cipher with an internal state of 160 bits $s_i, s_{i+1}, \ldots, s_{i+79}$ and $b_i, b_{i+1}, \ldots, b_{i+79}$ residing in a linear feedback shift register (LFSR) and a nonlinear feedback shift register (NLFSR), respectively. It supports a key $k = (k_0, \ldots, k_{79})$ of 80 bits and an initial value $IV = (IV_0, \ldots, IV_{63})$ of 64 bits. After a run-up time of 160 iteration steps it outputs a key stream z_i. During run-up the output bits z_i $(0 \leq i < 160)$ will not be used, but fed back into the LFSR and NLFSR components. Run-up (for $0 \leq i < 160$) and output generation (for $160 \leq i$) is described by the following recursion formula:

$$
\begin{aligned}
(b_0, \ldots, b_{79}) &:= (k_0, \ldots, k_{79}) \\
(s_0, \ldots, s_{79}) &:= (IV_0, \ldots, IV_{63}, 1, \ldots, 1) \\
g_i &:= g(b_{i+0}, b_{i+1}, \ldots, b_{i+63}) \\
f_i &:= s_{i+0} + s_{i+13} + s_{i+23} + s_{i+38} + s_{i+51} + s_{i+62} \\
\tilde{\sigma}_i &:= b_{i+1} + b_{i+2} + b_{i+4} + b_{i+10} + b_{i+31} + b_{i+43} \\
\sigma_i &:= \tilde{\sigma}_i + b_{i+56} \\
z_i &:= \sigma_i + h(s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, b_{i+63}) \\
b_{i+80} &:= g_i + s_i + z_i \cdot \delta_{[0,159]}(i) \\
s_{i+80} &:= f_i + z_i \cdot \delta_{[0,159]}(i)
\end{aligned}
$$

All variables represent elements of the binary field $\mathbb{F}_2$. $g\colon \mathbb{F}_2^{64} \to \mathbb{F}_2$ is a nonlinear function which we will not describe any further, since the exact form is not essential for the attack. The function h is defined by

$$\begin{aligned} h\colon \mathbb{F}_2^5 &\longrightarrow \mathbb{F}_2 \\ (x_0, \ldots, x_4) &\longmapsto x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + \\ &\qquad x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4. \end{aligned}$$

The indicator function $\delta_{[0,159]}(i)$ is 1 for $0 \le i \le 159$ and 0 otherwise. In addition to the specification in [3] we introduced the two intermediary values $\tilde{\sigma}_i$ and σ_i. They are not essential for the definition of the cipher. However, these values will be a part of our hypothesis.

Notation 1. *For the whole paper we will fix the secret key k. Besides k, the sequences b_i, s_i, g_i, f_i, $\tilde{\sigma}_i$, σ_i, z_i will depend on the initial value. Therefore we will often write $b_i^{(\nu)}$, $s_i^{(\nu)}$, ..., for $IV = \nu$.*

Remark 1. For $i = 0, \ldots, 16$, the elements $b_{i+63}^{(\nu)}$, $\sigma_i^{(\nu)}$, $\tilde{\sigma}_{i+17}^{(\nu)}$, $g_i^{(\nu)}$ do *not* depend on the initial value ν, but only on the key k.

2.2 Implementation in Hardware

A structural view of the hardware implementation is given in Fig. 2.

It consists of the following parts:

- an LFSR of 80 flip-flops $L_0, \ldots, L_{79}$, holding the values $s_i, \ldots, s_{i+79}$
- an NLFSR of 80 flip-flops $N_0, \ldots, N_{79}$, holding the values $b_i, \ldots, b_{i+79}$
- a combinatorial logic block G, realizing the function g
- a combinatorial logic block F, realizing the function f
- a combinatorial logic block H, realizing the function H
- some additional XORs.

Of course there is also some control logic for loading the key and initial value, clocking the FSRs, and switching δ.

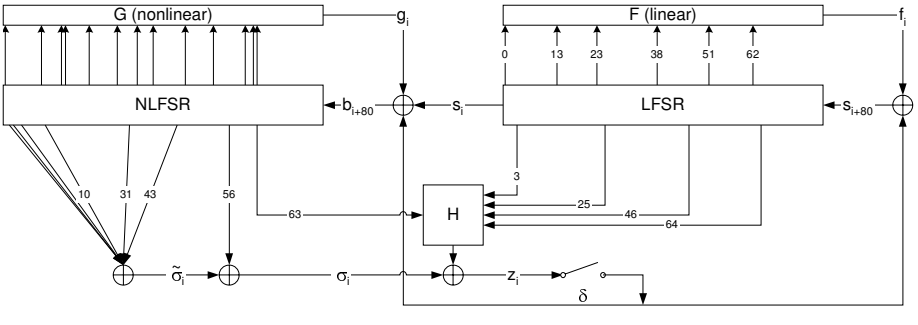

Fig. 2. Implementation of Grain

2.3 Power Model

We will use a discrete, Hamming distance based power model to describe the power consumption, since this suites the power consumption of a CMOS implementation very well. For a fixed key k and an initial value ν the power consumption of Grain is a function

$$P\colon \mathbb{N}_0 \longrightarrow \mathbb{R},$$

where $P(i)$ is the integral over the power consumption during the i-th clock cycle. The i-th clock cycle is the period of time, when the values g_i, f_i, $\tilde{\sigma}_i$, σ_i, z_i, b_{i+80}, s_{i+80} are evaluated and the two FSRs are shifted. Therefore we can write

$$P = P_{\mathrm{G}} + P_{\mathrm{H}} + P_{\mathrm{F}} + \sum_{j=0}^{79} P_{\mathrm{FF},N_j} + \sum_{j=0}^{79} P_{\mathrm{FF},L_j} + \Omega,$$

where P_{G}, P_{H}, P_{F} and P_{FF} denote the power consumption of G, H (including the generation of σ_i), F, and a flip-flop, respectively. Ω describes the noise which is independent of the described architectural elements. It is reasonable to model P_{G}, P_{H}, P_{F} and P_{FF} in a way, such that they only depend on the old and new input values:

$$\begin{aligned} P(i) = {} & P_{\mathrm{G}}(b_{i-1}, b_i, \ldots, b_{i+63}) \\ & + P_{\mathrm{H}}(b_i, \ldots, b_{i+63}, s_{i+2}, s_{i+3}, s_{i+24}, s_{i+25}, s_{i+45}, s_{i+46}, s_{i+63}, s_{i+64}) \\ & + P_{\mathrm{F}}(s_{i-1}, s_i, s_{i+12}, s_{i+13}, s_{i+22}, s_{i+23}, \ldots, s_{i+61}, s_{i+62}) \\ & + \sum_{j=0}^{79} P_{\mathrm{FF},N_j}(b_{i+j}, b_{i+j+1}) + \sum_{j=0}^{79} P_{\mathrm{FF},L_j}(s_{i+j}, s_{i+j+1}) + \Omega. \end{aligned}$$

Note, that this equation may not be fully correct for $i = 0$, since the "old" values may not always exist (e.g. b_{-1}) or could have some default values (after resetting the circuit). In this case the corresponding constant values must be used. We will make *no* further assumption about the functions $P_{\mathrm{G}}\colon \mathbb{F}_2^{65} \to \mathbb{R}$ and $P_{\mathrm{H}}\colon \mathbb{F}_2^{64+8} \to \mathbb{R}$, since this would add an unnecessary difficulty. It turns out that the precise form will not be needed. We define that $P_{\mathrm{F}}\colon \mathbb{F}_2^{12} \to \mathbb{R}$ is only a function of the Hamming distances of consecutive bits of the LFSR:

$$P_{\mathrm{F}}(s_{i-1}, s_i, \ldots, s_{i+61}, s_{i+62}) \equiv P_{\mathrm{F}}(s_{i-1} \oplus s_i, \ldots, s_{i+61} \oplus s_{i+62}).$$

For P_{FF} we make the usual approximation

$$P_{\mathrm{FF}}(0,0) \approx 0 \approx P_{\mathrm{FF}}(1,1) \ll P_{\mathrm{FF}}(1,0), P_{\mathrm{FF}}(0,1).$$

We do *not* assume that all P_{FF,N_j} or all P_{FF,L_j} are equal, as this cannot be expected to hold in an arbitrary implementation. Ω contains all noise contributions which are independent of the key and the initial value, such as the noise generated by the control hardware of the cipher or switching activity of circuits in the environment.

Notation 2. *For a fixed key k the whole cipher depends on the initial values ν. $P^{(\nu)}$, $P_G^{(\nu)}$, ..., denote the respective power consumption functions. $P^{(\nu)}$ will still be variable because of Ω. Therefore we will use its expectation value*

$$\bar{P}^{(\nu)} := \mathbb{E}P^{(\nu)},$$

which can be approximated by measuring the power consumption for the same initial value ν several times and taking the arithmetic mean value. Now $\bar{P}^{(\nu)}$ depends on the initial value ν only.

2.4 Attack on Grain: Theory

The attack on Grain consists of three steps. The first two steps are differential power analyses gaining information of 34 and 16 bits of the key, respectively. The third step is an exhaustive search on the remaining 30 bits of the key.

Step 1. is a DPA with chosen IVs. It is done in 17 rounds. In the i-th round ($0 \le i \le 16$) we set up our hypothesis (b_{i+63}^h, σ_i^h) about the pair (b_{i+63}, σ_i) and try to verify the true hypothesis by using the recorded power traces of the key setup phase for several initial values $\nu \in \mathcal{IV}_i$. The set $\mathcal{IV}_i$ of initial values is tailored in such a way, that the intrinsic power consumption of Grain will cancel out when computing the difference of the power traces (the "correlation function"). In each round the results of the previous ones will be used. Step 1 is illustrated in Table 1. Note that (b_{i+63}^h, σ_i^h) as well as ν must be used in order to

Table 1. DPA Attack on Grain, Step 1

For $i := 0$ **to** 16 **do**
  For all hypotheses $(b_{i+63}^h, \sigma_i^h) \in \mathbb{F}_2^2$ **do**
    Using hypothesis (b_{i+63}^h, σ_i^h) and the known $(b_{j+63}, \sigma_j)_{j=0}^{i-1}$, compute
    $\mathcal{IV}_i^+ := \{\nu \in \mathcal{IV}_i : s_{i+79}^{(\nu)} \neq s_{i+80}^{(\nu)}\}$, $\mathcal{IV}_i^- := \{\nu \in \mathcal{IV}_i : s_{i+79}^{(\nu)} = s_{i+80}^{(\nu)}\}$
    and the "correlation function"
    $$\bar{P}_{(b_{i+63}^h, \sigma_i^h)} := \frac{1}{\#\mathcal{IV}_i^+} \sum_{\nu \in \mathcal{IV}_i^+} \bar{P}^{(\nu)} - \frac{1}{\#\mathcal{IV}_i^-} \sum_{\nu \in \mathcal{IV}_i^-} \bar{P}^{(\nu)}$$
  end
  Accept the hypothesis, for which $\bar{P}_{(b_{i+63}^h, \sigma_i^h)}(i)$ is maximal.
end

compute the (hypothetical) value $s_{i+80}^{(\nu)}$. For computing $s_{i+79}^{(\nu)}$ the already known $(b_{i-1+63}, \sigma_{i-1})$ is used. The families $\mathcal{IV}_i$ (for $0 \le i \le 16$) of initial values are defined as follows:

$$\mathcal{IV}_i := \left\{ (\nu_0, \ldots, \nu_{63}) \in \mathbb{F}_2^{64} : \nu_n = \begin{cases} 0, & \text{for } n - i \neq 3, 13, 22, 23, 25, 46, \\ 1, & \text{for } n = i + 46. \end{cases} \right\}$$

$\mathcal{IV}_i$ contains 32 initial values which toggle the bits ν_{i+3}, ν_{i+13}, ν_{i+22}, ν_{i+23}, ν_{i+25} and set ν_{i+46} to 1.

Remark 2. In our case, we have $\#\mathcal{IV}_i^+ = \#\mathcal{IV}_i^- = 16$, for all $0 \le i \le 16$.

A justification for this algorithm and the choice of IVs is given in the following lemma.

Lemma 1. *In round i ($0 \le i \le 16$) of the above algorithm we have:*

$$\begin{aligned} P_{(b_{63},\sigma_0)}(0) &= \frac{1}{2}(P_{FF,L_{79}}(1,0) - P_{FF,L_{79}}(1,1)), \\ P_{(b_{i+63},\sigma_i)}(i) &= \frac{1}{2}(P_{FF,L_{79}}(0,1) + P_{FF,L_{79}}(1,0) \\ &\quad - P_{FF,L_{79}}(0,0) - P_{FF,L_{79}}(1,1)), \text{ for } 1 \le i, \\ P_{(b_{i+63},1+\sigma_i)}(i) &= -P_{(b_{i+63},\sigma_i)}(i), \\ P_{(1+b_{i+63},\sigma_i)}(i) &= P_{(1+b_{i+63},1+\sigma_i)}(i) = 0. \end{aligned}$$

For an explanation of this lemma cf. the paragraph after Lemma 2.

Remark 3. Of course, this DPA also works for families $\mathcal{IV}_i$ of randomly chosen initial values. However, the algorithmic noise level caused by the remaining 159 flip-flops (other than L_{79}) will be higher compared to the correlation signal. As a consequence the number of necessary samples $\#\mathcal{IV}_i$ for each family would be larger. The same is true for Step 2.

Step 2. works similarly to the first step, but only in 16 rounds. The hypotheses will be $(g^h_{i-17}, \tilde{\sigma}^h_i)$, for $17 \le i \le 32$. Moreover, the construction of the $\mathcal{IV}_i$ will make use of the previously learned data. The second step of the attack is given

Table 2. DPA Attack on Grain, Step 2

For $i := 17$ **to** 32 **do**
  For all hypotheses $(g^h_{i-17}, \tilde{\sigma}^h_i) \in \mathbb{F}_2^2$ **do**
    Using hypothesis $(g^h_{i-17}, \tilde{\sigma}^h_i)$ and the already known $(b_{j+63}, \sigma_j)_{j=0}^{16}$, $(g_{i-17}, \tilde{\sigma}_i)_{j=17}^{i-1}$, compute the partition of $\mathcal{IV}_i$
      $\mathcal{IV}_i^+ := \{\nu \in \mathcal{IV}_i : s_{i+79}^{(\nu)} \ne s_{i+80}^{(\nu)}\}$, $\mathcal{IV}_i^- := \{\nu \in \mathcal{IV}_i : s_{i+79}^{(\nu)} = s_{i+80}^{(\nu)}\}$
    and the "correlation function"
      $$\bar{P}_{(g^h_{i-17},\tilde{\sigma}^h_i)} := \frac{1}{\#\mathcal{IV}_i^+} \sum_{\nu \in \mathcal{IV}_i^+} \bar{P}^{(\nu)} - \frac{1}{\#\mathcal{IV}_i^-} \sum_{\nu \in \mathcal{IV}_i^-} \bar{P}^{(\nu)}$$
  end
  Accept the hypothesis, for which $\bar{P}_{(g^h_{i-17},\tilde{\sigma}^h_i)}(i)$ is maximal.
end

in Table 2. Note that $(g^h_{i-17}, \tilde{\sigma}^h_i)$, $(b_{j+63}, \sigma_j)_{j=0}^{16}$, $(g_{j-17}, \tilde{\sigma}_j)_{j=17}^{i-1}$ as well as ν must be used in order to compute the (hypothetical) value $s_{i+80}^{(\nu)}$. For computing $s_{i+79}^{(\nu)}$, the already known values $(b_{j+63}, \sigma_j)_{j=0}^{16}$, $(g_{j-17}, \tilde{\sigma}_j)_{j=17}^{i-1}$ are used. The families

$\mathcal{IV}_i$ (for $17 \le i \le 32$) of initial values are defined as follows:

$$\mathcal{IV}_i := \Big\{ \quad (\nu_0, \ldots, \nu_{63}) \in \mathbb{F}_2^{64} : (\nu_{i+3}, \nu_{i+13}, \nu_{i+22}, \nu_{i+23}, \nu_{i+25}) \in \mathbb{F}_2^5,$$

$$\nu_n = \begin{cases} \nu_{i+22}, & \text{if } n = i - 16 \wedge 16 \le i, \\ b_{i+44}, & \text{if } n = i + 6 \wedge 19 \le i, \\ 1, & \text{if } n = i + 27 \wedge 19 \le i < 37, \\ 0, & \text{if } n = i + 1 \wedge 24 \le i, \\ 1, & \text{if } n = i - 21 \wedge 24 \le i, \\ 1, & \text{if } n = 63 \wedge 17 = i, \\ 0, & \text{all other } n \in \{0, \ldots, 63\} \setminus \{3, 13, 22, 23, 25\} \end{cases} \Big\}.$$

Remark 4. Also in this case $\#\mathcal{IV}_i^+ = \#\mathcal{IV}_i^- = 16$, for all $17 \le i \le 32$.

The next lemma gives the justification for the algorithm.

Lemma 2. *In round i ($16 \le i \le 32$) of the above algorithm we have:*

$$\begin{aligned} P_{(g_{i-17}, \tilde{\sigma}_i)}(i) &= \frac{1}{2}(P_{FF,L_{79}}(0,1) + P_{FF,L_{79}}(1,0) \\ &\quad - P_{FF,L_{79}}(0,0) - P_{FF,L_{79}}(1,1)), \\ P_{(g_{i-17}, 1+\tilde{\sigma}_i)}(i) &= -P_{(g_{i-17}, \tilde{\sigma}_i)}(i), \\ P_{(1+g_{i-17}, \tilde{\sigma}_i)}(i) &= P_{(1+g_{i-17}, 1+\tilde{\sigma}_i)}(i) = 0. \end{aligned}$$

The proofs of the last two lemmata are straightforward, but involve rather lengthy calculations. They are preferably supported by an algebraic software package. The families of $\mathcal{IV}_i$ of initial value vectors are constructed such that the following properties hold:

- The toggling of the bits s_{i+13} and s_{i+23} directly causes a toggling of s_{i+80}, but not of s_{i+79}. This distributes the power functions $\bar{P}^{(\nu)}$, $\nu \in \mathcal{IV}_i$ to the two sums in the "correlation function" in a way, such that all power contributions not depending on s_{i+13} and s_{i+23} will cancel out.
- The toggling of the bit s_{i+22} has the same effect by influencing s_{i+79}.
- Function $h(x_0, x_1, x_2, x_3, x_4)$ has the following property: If $x_2 = x_3 = 1$ and $(x_0, x_1) \in \mathbb{F}_2^2$, changing x_4 to its complement results in a change of h in exactly 50% of the cases. This is the classical assumption for a bit to be used in a DPA.
- The additional conditions in the definition of $\mathcal{IV}_i$, for $17 \le i \le 32$ have the following reason. Toggling of bit s_{i+22} does not result in a toggling of other bits which also depend on the bits s_{i+13} and s_{i+23}, asserting that the first two facts are orthogonal.

Remark 5. In a practical attack on a hardware implementation the characteristics described in Lemmata 1 and 2 will transform into a peak for correct hypotheses (b_{i+63}, σ_i) or $(g_i, \tilde{\sigma}_i)$, and a negative peak for hypotheses with reversed σ_i or $\tilde{\sigma}_i$. Each of the other two hypotheses should not show a significant peak. During the following 79 clock cycles peaks can still be expected because the correlating signal remains.

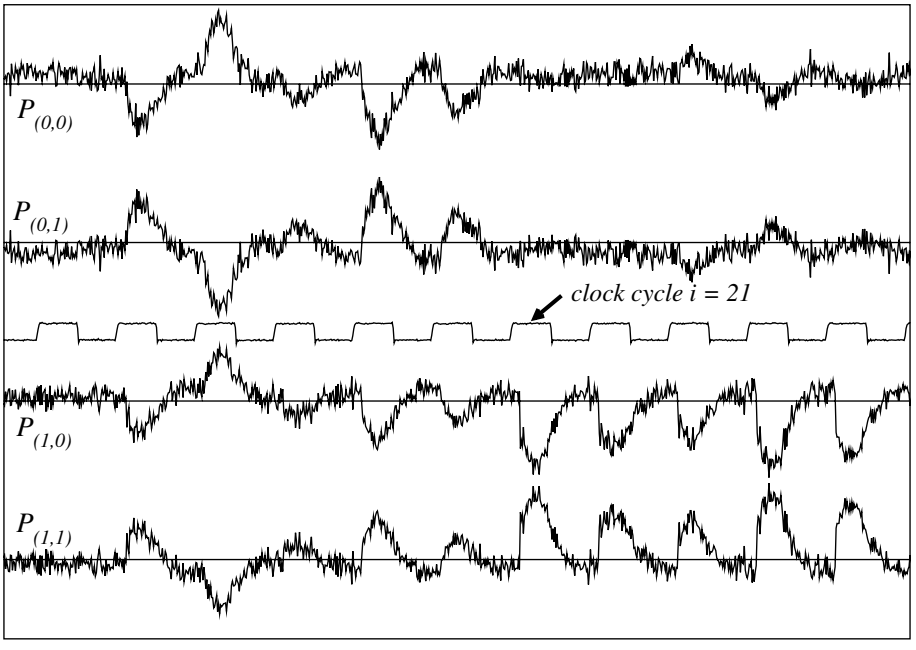

Fig. 3. The correlation functions $\bar{P}_{(0,0)}$, $\bar{P}_{(0,1)}$, $\bar{P}_{(1,0)}$, $\bar{P}_{(1,1)}$ for round 21 in the DPA attack on Grain

Step 3. is now straight forward. After having obtained 50 values $b_{63}, \ldots, b_{79}$, $\sigma_0, \ldots, \sigma_{16}$, $\tilde{\sigma}_{17}, \ldots, \tilde{\sigma}_{32}$, fifty independent linear equations in $k_0, \ldots, k_{79}$ can be written down. Solving these equations a linear map $\kappa\colon \mathbb{F}_2^{30} \to \mathbb{F}_2^{80}$ is obtained, such that the image of κ contains all possible remaining keys. Hence an exhaustive key search can be performed in practice. The complexity of this final step is $O(2^{30})$. The whole key can be recovered with, e.g., one appropriate plain text/cipher text pair. The attack can be improved by using the additional information contained in $(g_0, \ldots, g_{15})$ to exclude more keys.

2.5 Attack on Grain: Practical Realization

We implemented a version of Grain which generates one bit of key stream per clock cycle. This is probably the realization most relevant for hardware constrained environments. We chose an implementation on an Altera FLEX EPF10K100ARC240-2 FPGA as our target of attack. In a standard measurement setup the voltage drop at a shunt in the power supply line of the FPGA was measured. The FPGA was operated at 2.5 MHz and the power traces were recorded using a LeCroy LC684DXL oscilloscope with a sample rate of 2 Giga samples per second. A set of 256 power traces for each initial value in each family $\mathcal{IV}_i$ was obtained. The corresponding sample averages $\bar{P}^{(\nu)}$ were used to verify or falsify the hypotheses. As an example, in Figure 3 the four correlation

functions $\bar{P}_{(g^h_{i-17},\tilde{\sigma}^h_i)}$ for $i = 21$ and $(g^h_{i-17}, \tilde{\sigma}^h_i) \in \{(0,0),(0,1),(1,0),(1,1)\}$ are shown. As indicated by the arrow at clock cycle 21 the onset of peaks clearly verifies the correct hypothesis $(1,1)$.

3 Differential Power Analysis of Trivium

In this section we describe a DPA attack on the stream cipher Trivium [1]. This attack is not based on any nonlinear part, but correlations with the power consumption of the three flip-flops B_{81}, B_{82} and B_{83} are exploited. These flip-flops lie behind an XOR gate, which mixes known and controllable bits with secret bits. Again we are able to recover the whole key.

3.1 Definition of Trivium

Trivium is a stream cipher with an internal state of 288 bits $a_i, a_{i+1}, \ldots, a_{i+92}$, $b_i, b_{i+1}, \ldots, b_{i+83}$ and $c_i, c_{i+1}, \ldots, c_{i+110}$—residing in three coupled feedback shift registers A, B, and C of 93, 84, and 111 bits respectively—using a key $k = (k_0, \ldots, k_{79})$ of 80 bits as well as an initial value $IV = (IV_0, \ldots, IV_{79})$ of 80 bits. After a run-up time of $4 \cdot 288$ iteration steps it outputs a key stream z_i. Run-up (for $0 \le i < 4 \cdot 288$) and output generation (for $4 \cdot 288 \le i$) can be described by the following recursion formula:

$$\begin{aligned}
(a_0, \ldots, a_{92}) &:= (0, \ldots, 0, k_{79}, \ldots, k_0) \\
(b_0, \ldots, b_{83}) &:= (0, 0, 0, 0, IV_{79}, \ldots, IV_0) \\
(c_0, \ldots, c_{110}) &:= (1, 1, 1, 0, \ldots, 0) \\
a_{i+93} &:= a_{i+24} + c_i + c_{i+1}c_{i+2} + c_{i+45} \\
b_{i+84} &:= b_{i+6} + a_i + a_{i+1}a_{i+2} + a_{i+27} \\
c_{i+111} &:= c_{i+24} + b_i + b_{i+1}b_{i+2} + b_{i+15} \\
z_i &:= a_i + b_i + c_i + a_{i+27} + b_{i+15} + c_{i+45}
\end{aligned}$$

All variables represent elements in $\mathbb{F}_2$. The intermediate value $\sigma_i := a_i + a_{i+1}a_{i+2} + a_{i+27}$ will be our hypothesis in the DPA.

3.2 Implementation in Hardware

Again, the target of attack will be an implementation of the cipher which generates one bit of key stream per clock cycle. It comprises the following parts: An NLFSR of 93 flip-flops $A_0, \ldots, A_{92}$, holding the values $a_i, \ldots, a_{i+92}$, an NLFSR of 84 flip-flops $B_0, \ldots, B_{83}$, holding the values $b_i, \ldots, b_{i+83}$, an NLFSR of 111 flip-flops $C_0, \ldots, C_{110}$, holding the values $c_i, \ldots, c_{i+110}$, three additional AND, and a few XOR gates (as given in the recursion formula), as well as additional control logic for loading the key and initial value, and clocking the NLFSRs.

3.3 Power Model

We will use the same power model and notation as in the previous attack. The model for the power consumption is

$$P = \sum_{j=0}^{92} P_{\mathrm{FF},A_j} + \sum_{j=0}^{83} P_{\mathrm{FF},B_j} + \sum_{j=0}^{110} P_{\mathrm{FF},C_j} + \Omega,$$

as well as

$$P(i) = \sum_{j=0}^{92} P_{\mathrm{FF},A_j}(a_{i+j}, a_{i+j+1}) + \sum_{j=0}^{83} P_{\mathrm{FF},B_j}(b_{i+j}, b_{i+j+1}) + \sum_{j=0}^{110} P_{\mathrm{FF},C_j}(c_{i+j}, c_{i+j+1}) + \Omega.$$

Furthermore, we make the same assumption regarding the power consumption of the flip-flops as in the previous attack. To simplify the description we ignore the power consumption of the single AND and XOR gates. The notations, like $\bar{P}^{(\nu)}$, will also be used equivalently.

3.4 Attack on Trivium: Theory

To simplify the description we make the assumption that all flip-flops in our power model have the same power characteristic, i.e., for all appropriate j:

$$e_{xy} = P_{\mathrm{FF},A_j}(x,y) = P_{\mathrm{FF},B_j}(x,y) = P_{\mathrm{FF},C_j}(x,y)$$

with some constants $e_{00} \approx 0 \approx e_{11} \ll e_{01}, e_{10}$. This restriction, however, is not necessary to mount the attack.

The DPA is done in 76 rounds. In the i-th round we will know already $(\sigma_j)_{j=0}^{i-1}$ and evaluate σ_i. In fact, we will not need to make any hypothesis. The attack is

Table 3. DPA on Trivium

For $i := 3$ **to** 78 **except** 10 **do**
  Using the knowledge of $(\sigma_j)_{j=0}^{i-1}$, compute
    $\mathcal{IV}_i^+ := \{\nu_+\} := \{\nu \in \mathcal{IV}_i : b_{i+83}^{(\nu)} = 0\}$, $\mathcal{IV}_i^- := \{\nu_-\} := \{\nu \in \mathcal{IV}_i : b_{i+83}^{(\nu)} = 1\}$
  and
    $\bar{P}_i := \frac{1}{\#\mathcal{IV}_i^+} \sum_{\nu \in \mathcal{IV}_i^+} \bar{P}^{(\nu)} - \frac{1}{\#\mathcal{IV}_i^-} \sum_{\nu \in \mathcal{IV}_i^-} \bar{P}^{(\nu)}$
  if $b_{i+81}^{(\nu_+)} + b_{i+82}^{(\nu_+)} + b_{i+83}^{(\nu_+)} \equiv 0 \pmod 2$ **then**
    if $\bar{P}_i(i) > -(e_{01} + e_{10})/2$ **then** $\sigma_i := 1$ **else** $\sigma_i := 0$
  else
    if $\bar{P}_i(i) > (e_{01} + e_{10})/2$ **then** $\sigma_i := 1$ **else** $\sigma_i := 0$
end

illustrated in the following Table 3. We will assume, that the values σ_0, σ_1, σ_2, and σ_{10} are already known (in this case, one may make an "external" hypothesis on these 4 bits).[2]

The families $\mathcal{IV}_i$ (for $3 \le i \le 78$) of initial values are defined as follows:

$$\mathcal{IV}_i := \left\{(\nu_0, \ldots, \nu_{79}) \in \mathbb{F}_2^{80} : \nu_{78-i} = 1 + \nu_{79-i}, \nu_n := 0 \text{ otherwise}\right\}.$$

Note, that $\mathcal{IV}_i$ contains only 2 values. A justification for this algorithm and the choice of IVs is given in the following:

Lemma 3. *For any i, with $3 \le i \le 78$, $i \ne 10$, we have: (i) Writing $\mathcal{IV}_i = \{\nu_1, \nu_2\}$, then $b_{i+81}^{(\nu_1)} = b_{i+81}^{(\nu_2)}$ and $b_{i+82}^{(\nu_1)} + b_{i+83}^{(\nu_1)} \equiv b_{i+82}^{(\nu_2)} + b_{i+83}^{(\nu_2)} \pmod 2$, therefore the respective index "ν_+", in the above algorithm, can be left out. (ii) For $\bar{P}_i(i)$ we have the values:*

$(b_{i+81} + b_{i+82} + b_{i+83}) \mod 2$	b_{i+81}	$(b_{i+82} + b_{i+83}) \mod 2$	b_{i+84}	$\bar{P}_i(i)$		approx.
0	0	0	0	$2e_{00} - 2e_{11}$	$\approx$	0
0	0	0	1	$3e_{00} - e_{01} - e_{10} - e_{11}$	$\approx$	$-(e_{01} + e_{10})$
0	1	1	0	0	$\approx$	0
0	1	1	1	$e_{00} + e_{11} - e_{01} - e_{10}$	$\approx$	$-(e_{01} + e_{10})$
1	0	1	0	$e_{01} + e_{10} + e_{00} - 3e_{11}$	$\approx$	$(e_{01} + e_{10})$
1	0	1	1	$2e_{00} - 2e_{11}$	$\approx$	0
1	1	0	0	$e_{01} + e_{10} - e_{00} - e_{11}$	$\approx$	$(e_{01} + e_{10})$
1	1	0	1	0	$\approx$	0

Remark 6. In a practical attack on a hardware realization, by virtue of Lemma 3, the two inequalities will transform into the decisions {no peak↔negative peak} and {positive peak↔no peak}. The boundary $(e_{01} + e_{10})/2$ was just used for illustration purposes.

Extraction of the key: After gaining the 79 values $(\sigma_i)_{i=0}^{78}$ (possibly depending on the 4 hypothetical values σ_0, σ_1, σ_2, and σ_{10})we can write down the equations $\sigma_i = a_i + a_{i+1}a_{i+2} + a_{i+24}$ for $0 \le i \le 78$. These are 79 equations with 80 indeter-

$$\begin{array}{lll}
\sigma_0 = k_{65},\ \sigma_1 = k_{64} & \ldots & \sigma_{11} = k_{54}, \\
\sigma_{12} = k_{79}k_{78} + k_{53}, & & \\
\sigma_{13} = k_{79} + k_{78}k_{77} + k_{52}, & \ldots & \sigma_{65} = k_{27} + k_{26}k_{25} + k_0, \\
\sigma_{66} = k_{26} + k_{25}k_{24} + k_{68}, & & \\
\sigma_{67} = k_{25} + k_{24}k_{23} + k_{67} + 1, & & \\
\sigma_{68} = k_{24} + k_{23}k_{22} + k_{66}, & \ldots & \sigma_{78} = k_{14} + k_{13}k_{12} + k_{56},
\end{array}$$

minates $(k_i)_{i=0}^{79}$, which are shown explicitly in the following table. One equation is dependent on the others. For solving the system of equations, we may assume any value in $\mathbb{F}_2$ for k_{12} and k_{13}. By reordering the equations in the following Table —and leaving out the equation for σ_{12}—we can solve one equation after

[2] There is an other DPA strategy for evaluating $\sigma_0, \ldots, \sigma_{15}$. For lack of space we omit this evaluation phase.

$\sigma_0 = k_{65}$, $\sigma_1 = k_{64}$,	...	$\sigma_{11} = k_{54}$,	$\rightsquigarrow k_{65}, \ldots, k_{54}$
$\sigma_{27} = k_{65} + k_{64}k_{63} + k_{38}$,	...	$\sigma_{36} = k_{56} + k_{55}k_{54} + k_{29}$,	$\rightsquigarrow k_{38}, \ldots, k_{29}$
$\sigma_{54} = k_{38} + k_{37}k_{36} + k_{11}$,	...	$\sigma_{61} = k_{31} + k_{30}k_{29} + k_4$,	$\rightsquigarrow k_{11}, \ldots, k_4$
$\sigma_{78} = k_{14} + k_{13}k_{12} + k_{56}$,	...	$\sigma_{69} = k_{23} + k_{22}k_{21} + k_{65}$,	$\rightsquigarrow k_{14}, \ldots, k_{23}$
$\sigma_{53} = k_{39} + k_{38}k_{37} + k_{12}$,	...	$\sigma_{42} = k_{50} + k_{49}k_{48} + k_{23}$,	$\rightsquigarrow k_{39}, \ldots, k_{50}$
$\sigma_{26} = k_{66} + k_{65}k_{64} + k_{39}$,	...	$\sigma_{15} = k_{77} + k_{76}k_{75} + k_{50}$,	$\rightsquigarrow k_{66}, \ldots, k_{77}$
$\sigma_{68} = k_{24} + k_{23}k_{22} + k_{66}$,	...	$\sigma_{66} = k_{26} + k_{25}k_{24} + k_{68}$,	$\rightsquigarrow k_{24}, \ldots, k_{26}$
$\sigma_{41} = k_{51} + k_{50}k_{49} + k_{24}$,	...	$\sigma_{39} = k_{53} + k_{52}k_{51} + k_{26}$,	$\rightsquigarrow k_{51}, \ldots, k_{53}$
$\sigma_{38} = k_{54} + k_{53}k_{52} + k_{27}$,		$\sigma_{37} = k_{55} + k_{54}k_{53} + k_{28}$,	$\rightsquigarrow k_{27}, k_{28}$
$\sigma_{14} = k_{78} + k_{77}k_{76} + k_{51}$,		$\sigma_{13} = k_{79} + k_{78}k_{77} + k_{52}$,	$\rightsquigarrow k_{78}, k_{79}$
$\sigma_{62} = k_{30} + k_{29}k_{28} + k_3$,	...	$\sigma_{65} = k_{27} + k_{26}k_{25} + k_0$,	$\rightsquigarrow k_3, \ldots, k_0$

the other, getting a full key for each previously chosen pair (k_{12}, k_{13}). Counting also the hypotheses σ_0, σ_1, σ_2, and σ_{10} we may get at most $2^6 = 64$ different possible keys. Finding the right one is now trivial.

4 Conclusion

In this paper we showed, that DPA attacks on stream ciphers are practically feasible and that they constitute a real threat. We presented efficient differential power analyses of two new stream ciphers, which are focus candidates of the eSTREAM project. In both cases the DPA works with chosen IVs. These are carefully chosen to eliminate the algorithmic noise. It is plausible that this kind of attack can be applied to many stream ciphers with a similar construction philosophy.

References

1. Ch. De Cannière and B. Preneel: Trivium Specifications, 2005. Available at `http://www.ecrypt.eu.org/stream/p2ciphers/trivium/trivium_p2.pdf`.
2. S. Chari, J. R. Rao, and P. Rohatgi: Template Attacks. In B. S. Kaliski Jr., Ç. K. Koç, and Ch. Paar, editors, *Chryptographic Hardware and Embedded Systems – CHES 2002*, Lecture Notes in Computer Science, vol. 2535, pp. 13–28, Springer-Verlag, 2002.
3. M. Hell, Th. Johansson, and W. Meier: Grain – A Stream Cipher for Constrained Environments, 2006. Available at `http://www.ecrypt.eu.org/stream/p2ciphers/grain/Grain_p2.pdf`.
4. J. Hoch and A. Shamir: Fault Analysis of Stream Ciphers. In M. Joye and J.-J. Quisquater, editors, *Chryptographic Hardware and Embedded Systems – CHES 2004*, Lecture Notes in Computer Science, vol. 3156, pp. 240–253, Springer-Verlag, 2004.
5. P. C. Kocher, J. Jaffe, and B. Jun: Differential Power Analysis. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, Lecture Notes in Computer Science, vol. 1666, pp. 388–397, Springer-Verlag, 1999.
6. S. Kumar, K. Lemke, and Ch. Paar: Some Thoughts about Implementation Properties of Stream Ciphers. In *SASC 2004 – The State of the Art of Stream Ciphers*, (Brugge, Belgium, October 14-15, 2004), Workshop Record, pp. 311–319. Available at `http://www.ecrypt.eu.org/stvl/sasc/record.html`.

7. J. Lano, N. Mentens, B Preneel, and I. Verbauwhede: Power Analysis of Synchronous Stream Ciphers with Resynchronization Mechanism. In *SASC 2004 – The State of the Art of Stream Ciphers*, (Brugge, Belgium, October 14-15, 2004), Workshop Record, pp. 327–333. Available at `http://www.ecrypt.eu.org/stvl/sasc/record.html`.
8. J. Lano and G. Peeters: Cryptanalyse van NESSIE kandidaten (Dutch), Master's thesis, K. U. Leuven, May 2002.
9. Ch. Rechberger: Side Channel Analysis of Stream Ciphers, Master's thesis, Institute for Applied Information Processing and Communications (IAIK), Graz University, 2004.
10. Ch. Rechberger and E. Oswald: Stream Ciphers and Side-Channel Analysis. In *SASC 2004 – The State of the Art of Stream Ciphers*, (Brugge, Belgium, October 14-15, 2004), Workshop Record, pp. 320–326. Available at `http://www.ecrypt.eu.org/stvl/sasc/record.html`.

Cache Based Remote Timing Attack on the AES

Onur Acıiçmez[1], Werner Schindler[2], and Çetin K. Koç[1,3]

[1] Oregon State University, School of EECS
Corvallis, OR 97331, USA
{aciicmez,koc}@eecs.oregonstate.edu
[2] Bundesamt für Sicherheit in der Informationstechnik (BSI)
Godesberger Allee 185–189, 53175 Bonn, Germany
Werner.Schindler@bsi.bund.de
[3] Information Security Research Center, Istanbul Commerce University
Eminönü, Istanbul 34112, Turkey
koc@cryptocode.net

Abstract. We introduce a new robust cache-based timing attack on AES. We present experiments and concrete evidence that our attack can be used to obtain secret keys of remote cryptosystems if the server under attack runs on a multitasking or simultaneous multithreading system with a large enough workload. This is an important difference to recent cache-based timing attacks as these attacks either did not provide any supporting experimental results indicating if they can be applied remotely, or they are not realistically remote attacks.

Keywords: Cache Attack, Remote Attack, AES, Timing Analysis, Side Channel Analysis.

1 Introduction

The implementations of cryptosystems may leak information through so-called side channels due to the physical requirements of the device, e.g., power consumption, electromagnetic emanation and/or execution time. In side-channel attacks, the information obtained from one or more side-channels is used to reveal the key of a cryptosystem. Power, electromagnetic, and timing attacks are well-known types of side-channel attacks (c.f. [15,13,16,4,25]). Side-channel analysis of computer systems has recently attracted increasing attention (for a discussion see [3]). In this paper, we focus on a type of side channel cryptanalysis that takes advantage of the information leaks through the cache architecture of a CPU.

The feasibility of the cache based side channel attacks, abbreviated to "cache attacks" from here on, was first mentioned by Kocher and then by Kelsey et al. in [15,14]. D. Page described and simulated a theoretical cache attack on DES [21]. Cache attacks were first implemented by Tsunoo et al. [27,26]. They developed different attacks on various ciphers, including MISTY1 [26], DES and Triple-DES [27]. The recent efforts have unleashed the actual power of cache attacks [2,5,24,20,6,17,19,28,8].

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 271–286, 2007.

None of the mentioned papers, except [5], considered whether a remote cache attack is feasible. Although Bernstein claimed that his attack could reveal a full AES key remotely, his experiments were purely local [5] and he did not present sufficient evidence to support those claims. Furthermore, a thorough analysis of this attack showed that it could not compromise remote systems and could only recover the key partially in a local attack [18]

Despite of [7] the vulnerability of software systems against remote timing attacks was not taken into account until Brumley and Boneh performed an attack on unprotected SSL implementations over a local network ([10]). An improved version of this attack can be found in [1].

In this paper, we present a robust effective cache attack, which can be used to compromise remote systems, on the AES implementation described in [11] for 32-bit architectures. Although our basic principles can be used to develop similar attacks on other implementations, we will only focus on the particular implementation stated above.

Our paper is organized as follows: we will cover the basics of cache attacks in the next section. In Section 3, we will introduce our new cache attack on the AES. The results of the experiments will be presented along with the implementation details in Section 4. The paper ends with concluding remarks.

2 Basics of a Cache Attack

A cache is a small and fast storage area used by the CPU to reduce the average time to access main memory. It stores copies of the most frequently used data.[1] When the processor needs to read a location in main memory, it first checks to see if the data is already in the cache. If the data is already in the cache (a cache hit), the processor immediately uses this data instead of accessing the main memory, which has a longer latency than a cache. Otherwise (a cache miss), the data is read from the memory and a copy of it is stored in the cache. The minimum amount of data that can be read from the main memory into the cache at once is called a cache line or a cache block, i.e., each cache miss causes a cache block to be retrieved from a higher level memory.

Cache attacks exploit the cache hits and misses that occur during the encryption / decryption process of the cryptosystem. Even if the same instructions are executed for all (plaintext, cipherkey) pairs the cache behavior during the execution may cause variations in the program execution time and power consumption. Cache attacks try to exploit such variations to narrow the exhaustive search space of secret keys.

Theoretical cache attacks were first described by Page in [21]. Page characterized two types of cache attacks, namely trace-driven and time-driven. In trace-driven attacks (e.g. [2,6,17]), the adversary is able to obtain a profile of the cache activity of the cipher. This profile includes the outcomes of every memory access the cipher issues in terms of cache hits and misses. Therefore,

[1] Although it depends on the particular data replacement algorithm, this assumption is true almost all the time for current processors.

the adversary has the ability to observe (e.g.) if the 2^{nd} access to a lookup table yields a hit and can infer information about the lookup indices, which are key dependent. This ability gives an adversary the opportunity to make inferences about the secret key.

Time driven attacks, on the other hand, are less restrictive since they do not rely on the ability of capturing the outcomes of individual memory accesses [5,27,8]. The adversary is assumed to be able to observe the total execution time of the cipher, i.e. the aggregate profile, which at most gives hint to an approximative number of cache hits and misses, or to input-dependent correlations of particular operations. Time-driven attacks are based on statistical inferences, and therefore require much higher number of samples than trace-driven attacks.

We have recently seen another type of cache attacks that can be named as "access-driven" attacks [20,24,19]. In these attacks, the adversary can determine the cache sets that the cipher process modifies. Therefore, she can understand which elements of the lookup tables or S-boxes are accessed by the cipher. Then, the candidate keys that cause an access to unaccessed parts of the tables can be eliminated.

3 A New Remote Cache Attack on AES

All of the proposed cache attacks, except [5], either assume that the cache does not contain any data related to the encryption process prior to each encryption or explicitly force the cache architecture to replace some of the cipher data. The implementations of Tsunoo et al. accomplish the so-called 'cache cleaning' by loading some garbage data into the cache to clean it before each encryption [27,26]. The need of cleaning the cache makes an attack impossible to reveal information about the cryptosystems on remote machines, because the attacker must have an access to the computer to perform cache cleaning. They did not investigate if this attack could successfully recover the key without employing explicit cache cleaning on certain platforms.

Attacks described in [20] replace the cipher data on the cache with some garbage data by loading the content of a local array into the cache. Again, the attacker needs an access to the target platform to perform these attacks. Therefore, none of the mentioned studies could be considered as practical for remote attacks over a network, unless the attacker is able to manipulate the cache remotely.

In this paper, we show that it is possible to apply a cache attack without employing cache cleaning or explicitly aimed cache manipulations when the cipher under the attack is running on a multitasking system, especially on a busy server. In our experiments we run a dummy process simultaneously with the cipher process. Our dummy process randomly issues memory accesses and eventually causes the eviction of AES data from the cache. This should not be considered as a form of intentional cache cleaning, because we use this dummy process only to imitate a moderate workload on the server. In presence of different processes

that run on the same machine with the cipher process, the memory accesses that are issued by these processes automatically evict the AES data, i.e., cause the same effect of our dummy process on the execution of the cipher.

Multitasking operating systems allow the execution of multiple processes on the same computer, concurrently. In other words, each process is given permission to use the resources of the computer, not only the processor but also the cache and other resources. Although it depends on the cache architecture and the replacement policy, we can roughly say that the cache contains most recently used data almost all the time. If an encryption process stalls for enough time, the cipher data will completely be removed from the cache, in case of the presence of other processes on the machine. In a simultaneous multithreading system, the encryption process does not even have to stall. The data of the process, especially parts of large tables, is replaced by other processes' data on-the-fly, if there is enough workload on the system.

The results of our experiments show that the attack can work in such a case on a simultaneous multithreading environment. The reader should note that our results also point the vulnerability of remote systems against Tsunoo's attack on DES, as well.

In this section we outline an example cache attack on AES with a key size of 128 bits. In our experiments we consider the 128-bit AES version with a block length of 128 bits. Our attack can be adjusted to AES with key length 192 or 256 in a straight-forward manner (cf. Subsect. 3.4).

The basic attack consists of two different stages, considering table-lookups from the first and second round, respectively. The basic attack may be considered as an adaption of the ideas from the earlier cache attack works to a timing attack on AES since similar equations are used. Our improved attack variant is a completely novel approach. It employs a different decision strategy than the basic one and is much more efficient. It does not have different parts and falls into sixteen independent 8-bit guessing problems.

The differences of our approaches from the earlier works are the followings. First of all, we exploit the internal collisions, i.e., the collisions between different table lookups of the cipher. Some of the earlier works (e.g. [20,19,24,6]) exploits the cache collisions between the memory accesses of the cipher and another process. Exploiting such external collisions mandates the use of explicit local cache manipulations by (e.g.) having access to the target machine and reading a local data structure. This necessity makes these attacks unable to compromise remote systems. On the other hand, taking advantage of internal collisions removes this necessity and enables one to devise remote attacks as will be shown in this paper. The idea of using internal collisions is employed in some of the previous works, e.g. in [26,27,17]. The earlier timing attacks that rely on internal collisions perform the so-called cache cleaning, which is also a form of explicit local cache manipulations. These works did not realize the possibility of automatic cache evictions due to the workload on the system, and therefore could not show the feasibility of remote attacks.

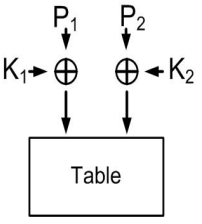

Fig. 1. Two different accesses to the same table

3.1 Basic Attack Model

We will use Figure 1 to explain the basic attack model. Assume there are two accesses to the same table. Let P_i and K_i be the i^{th} byte of the plaintext and cipherkey, respectively. In this paper, each byte is considered to be either an 8-digit radix-2 number $\in \{0,1\}^8$, that can be added in $GF(2^8)$ using a bitwise exclusive-or operation, or an integer in [0, 255] that can be used as an index. For the rest of this section, we assume that each plaintext consists of a single 16-byte message block.

The structure shown in the figure uses different bytes of the plaintext and the cipherkey as inputs to the function that computes the index of each of these two accesses. If both of them access to the same element of the table the latter should find the target data in the cache, resulting a cache hit; which should reduce the execution time. Then the key byte difference $K_1 \oplus K_2$ follows immediately from the values of plaintext bytes P_1 and P_2 using the equation

$$P_1 \oplus K_1 = P_2 \oplus K_2 \Rightarrow P_1 \oplus P_2 = K_1 \oplus K_2 \ .$$

In trace-driven attacks, we assume that the adversary can directly understand if the latter access results a hit, thus can directly obtain $K_1 \oplus K_2$. This goal is more complicated in time-driven attacks. We need to use a large sample to realize an accurate statistics of the execution. When sampling different plaintext pairs with the corresponding execution time we would expect that the plaintext byte difference $P_1 \oplus P_2$ that causes the shortest average execution time gives the correct key byte difference, assuming a cache hit decreases the overall execution time.

However, in a real environment the situation is more complicated. Even if the latter access is to a different element other than the target of the former access, a cache hit may still occur. Any cache miss results the transfer of an entire cache line, not only one element, from the main memory. Therefore, if the former access has a target, which lies in the same cache line of the previously accessed data, a cache hit will occur. In that case, we can still obtain the key byte difference partially as follows:

Let δ be the number of bytes in a cache line and assume that each element of the table is k bytes long. Under this situation, there are δ/k elements in each line, which means any access to a specific element will map to the same line with

$(\delta/k-1)$ different other accesses. If two different accesses to the same array read the same cache line, the most significant parts of their indices, i.e., all of the bits except the last $\ell = \log_2(\delta/k)$ bits, must be identical.[2] Using this fact, we can find the difference of the most significant part of the key bytes using the equation

$$\langle P_1 \rangle \oplus \langle P_2 \rangle = \langle K_1 \rangle \oplus \langle K_2 \rangle \ ,$$

where $\langle A \rangle$ stands for the most significant $(8-\ell)$ bits of A.

Indices of table lookups are driven by the outputs of usually more complex functions of the plaintext and the cipherkey than only bitwise exclusive-or of their certain bytes. The structure of these functions determines the performance of the attack, i.e., the amount of reduction in the exhaustive search space. The basic idea presented above can be adapted to any such function in order to develop successful attacks.

The attack model discussed so far is partially correct, except the lack of counting the fact that two different accesses to the same cache line may even increase the overall execution time. We realized during our experimentation that an internal collision, i.e. cache hit, at a particular AES access either shortens or lenghtens the overall execution time. The latter phenomenon may occur if a cache hit occurs from a logical point of view but the respective cache line has not already been loaded, inducing double work. Thus, if we gather a sample of messages and each of these messages generates a cache hit during the same access, then the execution time distribution of this sample will be significantly different than that of a random sample. We consider this fact to develop our attacks on the AES.

3.2 First Round Attack

The implementation we analyze is described in [11] and it is widely used on 32-bit architectures. To speed up encrytion all of the component functions of AES, except AddRoundKey, are combined into lookup tables and the rounds turn to be composed of table lookups and bitwise exclusive-or operations. The five lookup tables T0, T1, T2, T3, T4 employed in this implementation are generated from the actual AES S-box value as the following way:

$$T0[x] = (2 \bullet s(x), s(x), s(x), 3 \bullet s(x)), \quad T1[x] = (3 \bullet s(x), 2 \bullet s(x), s(x), s(x)),$$

$$T2[x] = (s(x), 3 \bullet s(x), 2 \bullet s(x), s(x)), \quad T3[x] = (s(x), s(x), 3 \bullet s(x), 2 \bullet s(x)),$$

$$T4[x] = (s(x), s(x), s(x), s(x)) \ ,$$

where s(x) and $\bullet$ stand for the result of an AES S-box lookup for the input value x and the finite field multiplication in $GF(2^8)$ as it is realized in AES, respectively. The round computations, except in the last round, are in the form:

$$(S^{(r+1)}_{(4*i)},\ S^{(r+1)}_{(4*i+1)},\ S^{(r+1)}_{(4*i+2)},\ S^{(r+1)}_{(4*i+3)}) := (RK^r_{(4*i)},\ RK^r_{(4*i+1)},\ RK^r_{(4*i+2)},\ RK^r_{(4*i+3)}) \oplus$$

[2] We assume that lookup tables are aligned in the memory, which is the case most of the time. If they are not aligned, this will indeed increase the performance of the attack as mentioned in [20].

$$T0[S^r_{(4*i)}] \oplus T1[S^r_{(4*i+5 \ mod \ 16)}] \oplus T2[S^r_{(4*i+10 \ mod \ 16)}] \oplus T3[S^r_{(4*i+15 \ mod \ 16)}] ,$$

where S^r_i is the byte i of intermediate state value that becomes the input of round r, RK^r_i is the i^{th} byte of the r^{th} round key and $i \in \{0, .., 3\}$.

The first 4 references to the first table, T0, are:

$$P_0 \oplus K_0, P_4 \oplus K_4, P_8 \oplus K_8, P_{12} \oplus K_{12} .$$

If any two of these four references are forced to map to the same cache line for a sample of plaintext then we know that this will affect the average execution time. For example, if we assign the value $\langle P_0 \oplus K_0 \oplus K_4 \rangle$ to $\langle P_4 \rangle$, i.e.,

$$\langle P_4 \rangle = \langle P_0 \oplus K_0 \oplus K_4 \rangle$$

for a large sample of plaintexts the timing characteristics of this sample will be different than that of a randomly chosen sample. We can use this fact to guess the correct key byte difference $\langle K_0 \oplus K_4 \rangle$.

Using the same idea we can find all key byte differences $\epsilon_{i,j} = \langle K_i \oplus K_j \rangle$ with $i, j \in \{0, 4, 8, 12\}$. For properly selected indices $(i_1, j_1), (i_2, j_2), (i_3, j_3)$, i.e. if the $GF(2)$-linear span of $\{K_{i_1} \oplus K_{j_1}, K_{i_2} \oplus K_{j_2}, K_{i_3} \oplus K_{j_3}\}$ contains all six XOR sums $K_0 \oplus K_4, K_0 \oplus K_8, \ldots, K_8 \oplus K_{12}$, then each $\epsilon_{i,j}$ follows immediately from $\epsilon_{i_1,j_1}, \epsilon_{i_2,j_2}$ and ϵ_{i_3,j_3}. We can further reduce the search space by considering the accesses to other three tables T1, T2 and T3. In general, we can obtain $\langle K_i \oplus K_{4*j+i} \rangle$ for $i, j \in \{0, 1, 2, 3\}$. Since $(8-\ell)$ is the size of the most significant part of a table entry in terms of the number of bits the first round attack allows us to reduce the search space by $12 * (8 - \ell)$ bits. The parameter ℓ depends on the cache architecture. For $\ell = 0$, which constitutes the theoretical lower bound, the search space for a 128 bit key becomes only 32 bits. For $\ell = 4$ the search space is reduced by 48 bits yielding an 80 bit problem.

On widely used processors the search space typically reduces to 56, 68, or 80 bits for 128-bit keys. In the environment where we performed our experiments the cache line size of the L1 cache is 64 bytes, i.e. the most significant part of a key byte difference is 4 bits long. In other words, we can only obtain the first 4 bits of $K_i \oplus K_{4*j+i}$ and the remaining 4 bits have to be searched exhaustively unless we use a second round attack.

3.3 Second Round Attack – Basic Variant

Using the guesses from the first round a similar guessing procedure can be applied in the second round to obtain the remaining key bits. We briefly explain an approach that exploits only accesses to T_0, i.e., the first table. To simplify notation we set $\Delta_i := P_i \oplus K_i$ in the remainder of this section. In the second round the encryption accesses four times to T0, namely to obtain the values

$$2 \bullet s(\Delta_8) \oplus 3 \bullet s(\Delta_{13}) \oplus s(\Delta_2) \oplus s(\Delta_7) \oplus s(K_{13}) \oplus K_0 \oplus K_4 \oplus K_8 \oplus 0x01 \quad (1)$$

$$2 \bullet s(\Delta_0) \oplus 3 \bullet s(\Delta_5) \oplus s(\Delta_{10}) \oplus s(\Delta_{15}) \oplus s(K_{13}) \oplus K_0 \oplus 0x01 \quad (2)$$

$$2 \bullet s(\Delta_4) \oplus 3 \bullet s(\Delta_9) \oplus s(\Delta_{14}) \oplus s(\Delta_3) \oplus s(K_{13}) \oplus K_0 \oplus K_4 \oplus 0x01 \quad (3)$$

$$2 \bullet s(\Delta_{12}) \oplus 3 \bullet s(\Delta_1) \oplus s(\Delta_6) \oplus s(\Delta_{11}) \oplus s(K_{13}) \oplus K_0 \oplus K_4 \oplus K_8 \oplus K_{12} \oplus 0x01 \quad (4)$$

where s(x) and • stand for the result of an AES S-box lookup for the input value x and the finite field multiplication in $GF(2^8)$ as it is realized in AES, respectively. If the first access $(P_0 \oplus K_0)$ touches the same cache line as (1) for each plaintext within a sample, i.e. if

$$\langle P_0 \rangle = \langle 2 \bullet s(\Delta_8) \oplus 3 \bullet s(\Delta_{13}) \oplus s(\Delta_2) \oplus s(\Delta_7) \oplus s(K_{13}) \oplus K_4 \oplus K_8 \oplus 0x01 \rangle \quad (5)$$

the expected average execution time will be different than for a randomly chosen sample. If we assume that the value $\langle K_4 \oplus K_8 \rangle$ has correctly been guessed within the first round attack this suggests the following procedure.

1. Phase: Obtain a sample of N many (plaintext, execution time) pairs.
2. Phase: Divide the entity of all (plaintext, execution time) pairs into 2^{32} (overlapping) subsets, one set for each candidate $(\widetilde{K_2}, \widetilde{K_7}, \widetilde{K_8}, \widetilde{K_{13}})$ value. Put each plaintext into all sets that correspond to candidates $(\widetilde{K_2}, \widetilde{K_7}, \widetilde{K_8}, \widetilde{K_{13}})$ that satisfy the above equation. Note that a particular plaintext should be contained in about $N/2^{8-\ell}$ different subsets.
3. Phase: Calculate the timing characteristics of each set, i.e., the average execution time in our case. Compute the absolute difference between each average and the average execution time of the entire sample. There will be a total of $2^{4 \cdot 8}$ timing differences, each from a different absolute value of $(\widetilde{K_2}, \widetilde{K_7}, \widetilde{K_8}, \widetilde{K_{13}})$. The set with the largest difference should point to the correct values for these 4 bytes.

Hence, we can search through all candidates for $(K_2, K_7, K_8, K_{13}) \in GF(2)^{32}$ to guess the true values. Applying the same idea to (2) to (4) we can recover the full AES key. Note that in each of the consecutive steps only $4 \cdot 4 = 16$ key bits have to be guessed since K_i and the most significant bits from some other K_j follow from the first step and ϵ_{ij} from the first round attack (cf. Sect. 3.2) where i is a suitable index in $\{2, 7, 8, 13\}$.

The bottleneck is clearly the first step since one has to distinguish between 2^{32} key hypotheses rather than between 2^{16}. Experimental results are given in Sect 4. In the next subsection we introduce a more efficient variant that saves both samples and computations.

3.4 A More Efficient, Universally Applicable Attack

In the previous subsection we explained a second round attack where 32, resp. 16, key bits have to be guessed simultaneously. In this section we introduce another approach that allows independent search for single key bytes. It is universally applicable in the sense that it could also be applied in any subsequent round, e.g. to attack AES with 256 bit keys.

We explain our idea at (1). Our goal is to guess key byte K_8. Recall that access to the same cache line as for $(P_0 \oplus K_0)$ is required in the second round iff (5) holds. If we fix the four plaintext bytes P_0, P_2, P_7, and P_{13} then (5) simplifies to

$$\langle c \rangle = \langle 2 \bullet s(\Delta_8) \rangle \quad (6)$$

with an unknown constant $c = c(K_0, K_2, K_4, K_7, K_8, K_{13}, P_0, P_2, P_7, P_{13})$. We observe encryptions with randomly selected plaintext bytes P_i for $i \notin \{0, 2, 7, 13\}$ and evaluate the timing characeristics with regard to all 256 possible values of P_8. For the most relevant case, i.e. $\ell = 4$, there are 16 plaintext bytes (2^ℓ in the general case) that yield the correct (but unknown) value $< 2 \bullet s(\Delta_8) >$ that meets (5). Ideally, with regard to the timing characteristics, these 16 plaintext bytes should be ranked first, pointing at the true subkey K_8; i.e. to a key byte that gives identical right-hand sides $< 2 \bullet s(\Delta_8) >$ for all these 16 plaintext bytes. The ranking is done similar as in Subsect. 3.2. To rank the 256 P_8-bytes one calculates for each subset with equal P_8 values the absolute difference of its average execution time with the average execution time of all samples. The set with the highest difference is ranked first and so on. In a practical attack our decision rule says that we decide for that key byte candidate $\widetilde{K}_8$ for which a maximum number of the t (e.g. $t = 16$) top-ranked plaintext bytes yield identical $\langle 2 \bullet s(\Delta_8) \rangle$ values. If the decision rule does not clearly point to one subkey candidate, we may perform the same attack with a second plaintext P_0' for which $\langle P_0 \rangle \neq \langle P_0' \rangle$ while we keep P_2, P_7, P_{13} fixed (changing $\langle c \rangle$ to $\langle c' \rangle := \langle c \rangle \oplus \langle P_0 \oplus P_0' \rangle$). Applying the same decision rule as above, we obtain a second ranking of the subkey candidates.

Clearly, if P_8 and P_8' meet (6) for P_0 and P_0', resp., then

$$\langle P_0 \oplus P_0' \rangle = \langle 2 \bullet s(P_8 \oplus K_8) \rangle \oplus \langle 2 \bullet s(P_8' \oplus K_8) \rangle. \tag{7}$$

Equation (7) may be used as a control equation for probable subkey candidates $\widetilde{K}_8$. From the ranking of $\widetilde{P}_8$ and $\widetilde{P}_8'$, we derive an order for the pairs $(\widetilde{P}_8, \widetilde{P}_8')$, e.g. by adding the ranks of the components or their absolute distances from the respective means. For highly ranked pairs $(\widetilde{P}_8, \widetilde{P}_8')$ we substitute $(\widetilde{P}_8, \widetilde{P}_8', \widetilde{k})$ into control equation (7) where $\widetilde{k}$ is a probable subkey candidate from the 'elementary' attacks.

We note that the attack described above can be applied to exploit the relation between any two table-lookups. By reordering a type (5)-equation one obtains an equation of type (6) whose right-hand side depends only on one key byte (to be guessed) and one plaintext byte. The plaintext bytes that affect the left-hand side are kept constant during the attack. The whole key could be recovered by 16 independent one-key byte guessing problems. We mention that the (less costly) basic first round attacks might be used to check the guessed subkey candidates $\widetilde{K}_0, \ldots, \widetilde{K}_{15}$.

Comparison with the Basic Second Round Attack from Subsect 3.3. For sample size N the 'bottleneck' of the basic second round attack, the 32 bit guessing step, requires the computation of the average execution times of 2^{32} sample subsets of size $\approx N/2^{8-\ell}$. In contrast, each of the 16 runs of the improved attack variant only requires the computation of the average execution times of 256 subsets of size $\approx N_I/256$ (with N_I denoting the sample size for an individual guessing problem) and sorting two lists with 256 elements (plaintexts and key byte candidates). Even more important, $16N_I$ will turn out to be clearly smaller than N (cf. Sect. 4).

The only drawback of the improved variant is that it is a chosen-input attack, i.e., it requires an active role of the adversary. In contrast, the basic variant explained in the previous section is principally a known-plaintext attack, which means an adversary does not have to actively interfere with the execution of the encryption process, i.e., the attack can be carried out by monitoring the traffic of the encryption process. However, this is only true for the (less important) so-called innerprocess attacks (cf. Section 4 for details). For 'real' attacks (interprocess and remote attacks) the basic variant is performed as a chosen-input attack, too, since the attacker needs to choose the plaintext to be encrypted as the concatenation of several identical 128 bit strings in order to increase the signal-to-noise ratio.

4 Experimental Details and Results

We performed two types of experimental attacks that we call innerprocess and interprocess attacks to test the validity of our attack variants. In innerprocess attack we generated a random single-block of messages and measured their encryption times under the same key. The encryption was just a function that is called by the application to process a message. The execution time of the cryptosystem was obtained by calculating the difference of the time just before the function call and immediately after the function return. Therefore, there was minimum noise and the execution time was measured almost exactly.

For the second part of the experiments, i.e., interprocess attack, we implemented a simple TCP server and a client program that exchange ASCII strings during the attack. The server reads the queries sent by the client, and sends a response after encrypting each of them. The client measures the time between sending a message and receiving the reply. These measurements were used to guess the secret key. The server and client applications run on the same machine in this attack. There was no transmission delay in the time measurements but network stack delays were present.

Brumley and Boneh pointed out that a real remote attack over a network was principally able to break a remote cipher, when the interprocess version of the same attack worked successfully. Furthermore, their experiments also showed that their actual remote attack required roughly the same number of samples used in the interprocess version [10]. Therefore, we only performed interprocess experiments. Applying an interprocess attack successfully is a sufficient evidence to claim the actual remote version would also work with (most likely) a larger sample size.

We performed our attack against OpenSSL version 0.9.7e. All of the experiments were run on a 3.06 GHz. HT-enabled Xeon machine with a Linux operating system. The source code was compiled using the gcc compiler version 3.2.3 with default options. We used random plaintexts generated by rand() and srand() functions available in the standard C library. The current time is fed into srand() function, serving as seed for the pseudorandom number generator. We measured time in terms of clock cycles using the cycle counter.

For the experiments of innerprocess attack, we loaded 8 KB garbage data into the L1 cache before each encryption to remove all AES data from the first level

cache. We did not employ this type of cache cleaning during the experiments of interprocess attack. Instead, we wrote a simple dummy program that randomly accesses an 8 KB array and run this program simultaneously with the server in order to imitate the effect of a workload on the computer.

We used two parameters in our experiments.

1. Sample Size (N): This is the number of different (plaintext, execution time) pairs collected during the first phase of the attacks. We have to use a large enough sample of queries to obtain accurate statistical characteristic of the system. However, a very large sample size causes unnecessary increase in the cost of the attack.
2. Message Length (L): This is the number of message blocks in each query. We concatenated a single random block L many times with one another to form the actual query. L was 1 during the innerprocess attack, i.e., each query was a single block, whereas it was 1024 in the interprocess attack. This parameter is used to increase the signal-to-noise ratio in the case of having network delays in the measurements.

We performed our attacks on the variant of AES that has 128-bit key and block sizes. The cache line size of L1 cache is 64 bytes, which makes $\ell = 4$ bits. The cipher was run on ECB mode. In our experiments, we performed all second round guessing problems for the basic attack with only 2^{12} different key hypotheses, one of them being the correct key combination. Our intention was to demonstrate the general principle but to save many encryptions. In this way, we reduced the complexity of 'bottleneck' exhaustive search by even more than a factor of 2^{20} since less samples are sufficient for the reduced search space.

For the innerprocess attack, collecting 2^{18} samples was enough to find the correct value of the key. Since we only considered 2^{12} different key hypotheses in second round guessing problems, the required sample size would be more than 2^{18} for a real scale innerprocess attack. In fact, statistical calculations suggest that $4*2^{18}$ samples should be sufficient for 2^{32} key hypotheses although in a strict sense (13) only guarantees an error probability of at most $2\epsilon/(1-c)-\epsilon^2/(1-c)^2 > 2\epsilon - \epsilon^2$ (cf. Example 1 in the appendix). (The right-hand side denotes the error probability for the reduced search space while c is unknown.) However, (11) is a (pessimistic) lower bound we may expect that the true error probability is indeed significantly smaller, possibly after increasing the sample size somewhat.

The key experiment is the interprocess attack, which shows the vulnerability of remote servers against such cache attacks. In our experiments, we collected 50 million random but known samples and applied our attack on this sample set. This sample size was clearly sufficient to reveal the correct key value among 2^{12} different key hypotheses. Again, the same heuristic arguments indicate the sufficiency of 200 million samples in a real-scale attack. We also estimated the number of required samples in a remote attack over a local network. Rough statistical considerations indicate that increasing the sample size of the interprocess attack by a factor of less than 6 should be sufficient to successfully apply the attack on a remote server.

We tested our improved variant on the same platform with the same settings. The only difference was the set of the plaintexts sent to the server. We only performed interprocess attack with this new decision strategy. Our experimental results indicate a clear improvement over the basic attack. We could recover a full 128-bit AES key by encrypting slightly more than 6.5 million samples in average per each of the 16 guessing problems and a total of 106 million queries, each containing $L = 1024$ message blocks. Recall the further advantage of the improved variant, namely the much lower analysis costs.

We want to mention that all of these results correspond to the minimum number of samples from which we got the correct decision from our decision strategy. In a real-life-attack an adversary clearly has to collect more samples to be confident on her decisions in a real attack. More sophisticated stochastic models that are tailored to specific cache strategies certainly will improve the efficiency of our attack.

Our client-server model does not perfectly fit into the behavior of an actual security application. In reality, encrypting/decrypting parties do not send responses immediately and perform extra operations, besides encryption and decryption. However, this fact does not nullify our client-server model. Although, the less signal-to-noise ratio in actual attacks increases the cost, it does not change the principle feasibility of our attacks. We want to mention that timing variations caused by extra operations decrease the signal-to-noise ratio. If a security application performs the same operations for each processed message, we expect the "extra timing variations" to be minimal, in which case the decrease in the signal-to-noise ratio and thus the increase in the cost of the attack also remains small.

5 Conclusion

We have presented a new cache-based timing attack on AES software implementations. Our experiments indicate that cache attacks can be used to extract secret keys of remote systems if the system under attack runs on a server with a multitasking or multithreading system and a large enough workload. Although a large number of measurements are required to successfully perform a remote cache attack, it is feasible in principle. In this regard, we would like to point the feasibility of such cache attacks to the public, and recommend implementing appropriate countermeasures. Several countermeasures [21,5,20,22,23,9] have been proposed to prevent possible vulnerabilities and develop more secure systems.

References

1. O. Acıiçmez, W. Schindler, Ç. K. Koç. Improving Brumley and Boneh Timing Attack on Unprotected SSL Implementations. ACM CCS'05, C. Meadows, P. Syverson, editors, 139-146, Virginia, 2005.
2. O. Acıiçmez and Ç. K. Koç. Trace-Driven Cache Attacks on AES. Cryptology ePrint Archive, Report 2006/138, 2006. Available at: http://eprint.iacr.org/2006/138

3. O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert. Predicting Secret Keys via Branch Prediction. Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, to appear.
4. D. Agrawal, B. Archambeault, J. R. Rao, P. Rohatgi. The EM Side-Channel(s). CHES'02, B. S. Kaliski, Ç. K. Koç and C. Paar, editors, 29-45, Springer, LNCS 2523, Berlin 2003.
5. D. J. Bernstein. Cache-timing attacks on AES. April, 2005. Available at: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf
6. G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, G. Palermo. AES Power Attack Based on Induced Cache Miss and Countermeasure. ITCC'05, 586 - 591, IEEE Computer Society, 2005.
7. D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. CRYPTO 98, H. Krawczyk, editor, 1-12, Springer, LNCS 1462, Berlin 1998.
8. J. Bonneau and I. Mironov. Cache-Collision Timing Attacks against AES. CHES'06, Springer, LNCS , Berlin 2006.
9. E. Brickell, G. Graunke, M. Neve, J.-P. Seifert. Software mitigations to hedge AES against cache-based software side channel vulnerabilities. Cryptology ePrint Archive, Report 2006/052, 2006. Available at: http://eprint.iacr.org/2006/052
10. D. Brumley, D. Boneh. Remote Timing Attacks are Practical. Proceedings of the 12^{th} Usenix Security Symposium, 1-14, 2003.
11. J. Daemen, V. Rijmen. "The Design of Rijndael". Springer, Berlin 2001.
12. W. Feller. An Introduction to Probability Theory. 3^{rd} edition, revised printing, Wiley, New York 1970.
13. K. Gandolfi, C. Mourtel, F. Olivier. Electromagnetic Analysis: Concrete Results. CHES'01, Ç. K. Koç, D. Naccache, and C. Paar, editors, 251-261, Springer, LNCS 2162, Berlin 2001.
14. J. Kelsey, B. Schneier, D. Wagner, C. Hall. Side Channel Cryptanalysis of Product Ciphers. Journal of Computer Security, vol.8, 141-158, 2000.
15. P. C. Kocher. Timing Attacks on Implementations of Diffie–Hellman, RSA, DSS, and Other Systems. CRYPTO '96, N. Koblitz, editor, 104-113, Springer, LNCS 1109, Berlin 1996.
16. P. C. Kocher, J. Jaffe, B. Jun. Differential Power Analysis. CRYPTO '99, M. Wiener, editor, 388-397, Springer, LNCS 1666, Berlin 1999.
17. C. Lauradoux. Collision attacks on processors with cache and countermeasures. WEWoRC'05, C. Wolf, S. Lucks, and P.-W. Yau, editors, 76-85, KÏlen, LNI P-74, Bonn 2005.
18. M. Neve, J.-P. Seifert, Z. Wang. A refined look at Bernstein's AES side-channel analysis. ASIA CCS'06, 369-369, ACM Press, 2006.
19. M. Neve and J.-P. Seifert. Advances on Access-driven Cache Attacks on AES. SAC'06, E. Biham, A. Youssef, editors, to appear.
20. D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. CT-RSA'06, D. Pointcheval, editor, 1-20, Springer, LNCS 3860, Berlin 2006.
21. D. Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, June 2002.
22. D. Page. Defending Against Cache Based Side-Channel Attacks. Technical Report. Department of Computer Science, University of Bristol, 2003.

23. D. Page. Partitioned Cache Architecture as a Side Channel Defence Mechanism. Cryptography ePrint Archive, Report 2005/280, 2005. Available at: http://eprint.iacr.org/2005/280
24. C. Percival. Cache missing for fun and profit. BSDCan'05, Ottawa, 2005. Available at: http://www.daemonology.net/hyperthreading-considered-harmful/
25. W. Schindler: On the Optimization of Side-Channel Attacks by Advanced Stochastic Methods, PKC '05, S. Vaudenay, editor, 85–103, Springer, LNCS 3386, Berlin 2005.
26. Y. Tsunoo, E. Tsujihara, K. Minematsu, H. Miyauchi. Cryptanalysis of Block Ciphers Implemented on Computers with Cache. ISITA'02, 803-806, IEEE Information Theory Society, 2002.
27. Y. Tsunoo, T.Saito, T. Suzaki, M. Shigeri, H. Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. CHES'03, C. D. Walter, Ç. K. Koç, and C. Paar, editors, 62-76, Springer, LNCS 2779, Berlin 2003.
28. Y. Tsunoo, E. Tsujihara, M. Shigeri, H. Kubo, K. Minematsu. Improving cache attacks by considering cipher structure. International Journal of Information Security, February 2006.

Appendix: Scaling the Sample Size N

In order to save measurements we performed our practical experiments to the basic second round attack from Subsect. 3.3 with a reduced key space. Clearly, to maintain the success probability for the full subkey space the sample size N must be increased to N' since the adversary has to distinguish between more admissible alternatives. In this section we estimate the ratio $r := N'/N$.

We interpret the measured average execution times for the particular subkey candidates as realizations of normally (Gaussian) distributed random variables, denoted by Y (related to the correct subkey) and $X_1, \ldots, X_{m-1}$ (related to the wrong subkey candidates) for the reduced subkey space, resp. $X_1, \ldots, X_{m'-1}$ when all possible subkeys are admissible. We may assume $Y \sim N(\mu_A, \sigma_A^2)$ while $X_j \sim N(\mu_B, \sigma_B^2)$ for $j \leq m-1$, resp. for $j \leq m'-1$, with unknown expectations μ_A and μ_B and variances σ_A^2 and σ_B^2. Clearly, $\mu_A \neq \mu_B$ since our attack exploits differences of the average execution times. Since it only exploits the relation between two table lookups $\sigma_A^2 \approx \sigma_B^2$ seems to be reasonable, the variances clearly depending on N. W.l.o.g. we may assume $\mu_A > \mu_B$. We point out that $E(X_1 + \ldots + X_{m-1} + Y)/m \approx \mu_B$ unless m is very small.

$$\begin{aligned} &\mathrm{P}rob(\text{correct guess}) \approx \mathrm{P}rob(|Y - \mu_B| > \max\{|X_1 - \mu_B|, \ldots, |X_{m-1} - \mu_B|\}) \\ &= \mathrm{P}rob(\min\{X_1, \ldots, X_{m-1}\} > \mu_B - (Y - \mu_B), \max\{X_1, \ldots, X_{m-1}\} < Y) \\ &\approx \mathrm{P}rob(\max\{X_1, \ldots, X_{m-1}\} < Y)^2 \end{aligned} \tag{8}$$

Unless m is very small the $\approx$ sign should essentially be "=". If the random variables $Y, X_1, \ldots, X_{m-1}$ were independent we had

$$\begin{aligned} \mathrm{P}rob(\max\{X_1, \ldots, X_{m-1}\} \leq t) &= \prod_{j=1}^{m-1} \mathrm{P}rob(X_j \leq t) = \\ &= \Phi((t - \mu_B)/\sigma_B)^{m-1} \end{aligned} \tag{9}$$

where Φ denotes the cumulative distribution function of the standard normal distribution. From (9) one immediately deduces

$$Prob(\max\{X_1, ..., X_{m-1}\} < Y) \approx \int_{-\infty}^{\infty} \Phi((z-\mu_B)/\sigma_B)^{m-1} f_A(z)dz \qquad (10)$$

where Y has density f_A. In the context of Subsect. 3.3 the random variables $Y, X_1, \ldots, X_{m-1}$ are yet dependent. However, for different subkey candidates k_i and k_j the size of the intersection of the respective subsets is small compared to the size of these subsets themselves. Hence we may hope that the influence of the correlation between X_i and X_j is negligible. Under this asumption (10) provides a concrete formula for the probability for a true guess. However, this formula cannot be evaluated in practice since μ_A, μ_B and $\sigma_A^2 \approx \sigma_B^2$ are unknown. Instead, we prove useful Lemma.

Lemma 1. *(i) Let f denote a probability density, while $0 \le g, h \le 1$ are integrable functions and $M_c := \{y : g(y) \le c\}$. Assume further that $h \ge g$ on $R \setminus M_c$. Then*

$$\int h(z)f(z)dz \ge 1 - \frac{\epsilon}{1-c} \quad \text{if} \quad \int g(z)f(z)dz = 1 - \epsilon \qquad (11)$$

(ii) Let $s, u, b > 1$. Then there exists a unique $y_0 > 0$ with $\Phi(y_0 s)^{ub} = \Phi(y_0)^b$. In particular, $\Phi(ys)^{ub} > \Phi(y)^b$ iff $y > y_0$.

Proof. Assertion (i) follows immediately from

$$(1-c)\int_{M_c} f(z)dz \le \int_{M_c} (1-g(z))f(z)dz \le \int (1-g(z))f(z)dz = \epsilon$$

and hence

$$\begin{aligned}\int h(z)f(z)\,dz &\ge 0 + \int_{\mathrm{R}\setminus M_c} g(z)f(z)\,dz = (1-\epsilon) - \int_{M_c} g(z)f(z) \\ &\ge (1-\epsilon) - c\int_{M_c} f(z)\,dz \ge 1 - \epsilon - \frac{c\epsilon}{1-c} = 1 - \frac{\epsilon}{1-c}.\end{aligned}$$

Since $\Phi(ys)^{ub}/\Phi(y)^b = \left(\Phi(ys)^u/\Phi(y)\right)^b$ we may assume $b = 1$ in the remainder w.l.o.g. Clearly, $\Phi(ys)^u < \Phi(y)$ for $y < 0$. Hence we concentrate to the case $y \ge 0$. In particular, $\log(1-x) = -x + O(x^2)$ implies

$$\begin{aligned}\psi(y) &:= \log\left(\Phi(ys)^u/\Phi(y)\right) = u\log(\Phi(ys)) - \log(\Phi(y)) \\ &= u\log(1-(1-\Phi(ys))) - \log(1-(1-\Phi(y))) \\ &= -u\,(1-\Phi(ys)) + (1-\Phi(y)) + O\left((1-\Phi(y))^2\right) \\ &\ge \frac{1}{\sqrt{2\pi}}\left(\frac{1}{y} - \frac{1}{y^3}\right)e^{-y^2/2} - \frac{1}{\sqrt{2\pi}}\frac{u}{ys}\left(e^{-y^2/2}\right)^{s^2} + O\left(\left(\frac{1}{y}e^{-y^2/2}\right)^2\right) \\ &> 0 \text{ for sufficiently large } y, \qquad \text{and } \lim_{y\to\infty}\psi(y) = 0. \end{aligned} \qquad (12)$$

We note that the last assertion follows immediately from the definition of ψ while the '$\ge$' sign is a consequence from a well-known inequality of the tail of

$1-\Phi$ (see, e.g., [12], Chap. VII, 175 (1.8)). Since ψ is continuous and $\psi(0) = \log(0.5^{u-1}) < 0$ there exists a minimal $y_0 > 0$ with $\psi(y_0) = 0$. For any $y_1 \in \{y \geq 0 \mid \psi'(y) = 0\}$ the second derivative simplifies to $\psi''(y_1) = t(y_1)\Phi'(y_1)/\Phi(y_1)$ with $t(x) := (1-s^2)x + (1-1/u)\Phi'(x)/\Phi(x)$. (Note that $\Phi''(ys) = -ys\Phi'(ys)$ and $\Phi''(y) = -y\Phi'(y)$.) Assume that $\psi(y_{00}) = 0$ for any $y_{00} > y_0$. As $\psi(0) < 0$ and $\psi(y_0) = \psi(y_{00}) = 0$ the function ψ attains a local maximum in some $y_m \in [0, y_{00})$. Since $t\colon [0,\infty) \to \mathrm{R}$ is strictly monotonously decreasing ψ cannot attain a local minimum in (y_m, ∞) (with $\psi(\cdot) \leq 0 = \psi(y_{00})$) which contradicts (12). This proves the uniqueness of y_0 and completes the proof of (ii).

Our goal is to apply Lemma 1 to the right-hand side of (10). We set $u := (m'-1)/(m-1)$, $b := 1$ and $s := \sqrt{r}$ with $r := N'/N$. Further, $f(z) := f_A(z)$, $g(z) := (\Phi((z-\mu_B)/\sigma_B))^{m-1}$ and $h(z) := (\Phi(\sqrt{r}(z-\mu_B)/\sigma_B))^{u(m-1)}$. By (ii) we have $c = \Phi((z_0-\mu_B)/\sigma_B)^{m-1}$ and $M_c = (\infty, z_0]$ with $g(z_0) = h(z_0)$. Lemma 1 and (8) imply

$$\left[\mathrm{P}rob(\text{correct guess for } (m,N)) = (1-\epsilon)^2\right] \Rightarrow \tag{13}$$
$$\left[\mathrm{P}rob(\text{correct guess for } (m', N' = rN)) \geq \left(1 - \frac{\epsilon}{1-c}\right)^2\right]$$

providing a lower probability bound for a correct guess in the full key space attack. Note that $\mu_A, \mu_B, \sigma_A^2 \approx \sigma_B^2, N, r$ determine ϵ, c and z_0 which are yet unknown in real attacks since μ_A and μ_B are unknown. Example 1 gives an idea of the magnitude of r.

Example 1. Let $m = 2^{12}$, $m' = 2^{32}$, and $y_0 := (z_0-\mu_B)/\sigma_B = \Phi^{-1}(c^{1/(m-1)})$. If $c = 0.5$ (resp., if $c = 100/101$) the number $r = N'/N = 3.09$ (resp., $r = 3.85$) gives $\Phi(y_0\sqrt{r})^{u(m-1)} = \Phi(y_0)^{m-1} = 0.5$ (resp., $= 100/101$).

Group Secret Handshakes Or Affiliation-Hiding Authenticated Group Key Agreement

Stanisław Jarecki, Jihye Kim, and Gene Tsudik

Computer Science Department
University of California, Irvine
{stasio, jihyek, gts}@ics.uci.edu

Abstract. Privacy concerns in many aspects of electronic communication trigger the need to re-examine – with privacy in mind – familiar security services, such as authentication and key agreement.

An *Affiliation-Hiding* Group Key Agreement (AH-AGKA) protocol (also known as *Group Secret Handshake*) allows a set of participants, each with a certificate issued by the same authority, to establish a common authenticated secret key. In contrast to standard AGKA protocols, an AH-AGKA protocol has the following privacy feature: If Alice, who is a member of a group G, participates in an AH-AGKA protocol, none of the other protocol participants *learn* whether Alice is a member of G, unless these participants are themselves members of group G. Such protocols are useful in suspicious settings where a set of members of a (perhaps secret) group need to authenticate each other and agree on a common secret key, without revealing their affiliations to outsiders.

In this paper we strengthen the prior definition of AH-AGKA so that the security and privacy properties are maintained under any composition of protocol instances. We also construct two novel AH-AGKA protocols secure in this new and stronger model under the RSA and Gap Diffie-Hellman assumptions, respectively. Each protocol involves only two communication rounds and few exponentiations per player (e.g., no bilinear map operations). Interestingly, these costs are essentially the same as those of the underlying (*unauthenticated*) group key agreement protocol. Finally, our protocols, unlike prior results, retain their security and privacy properties without the use of one-time certificates.

Keywords: secret-handshakes, group key agreement, authenticated group key agreement, privacy, privacy-preserving authentication.

1 Introduction

A traditional authenticated group key agreement (AGKA) protocol is assumed to operate within the confines of a common Public Key Infrastructure (PKI). At the start, participants – who have no prior secrets in common – exchange their public key certificates (PKCs). This exchange leaks information; in particular, it always reveals a participant's public key certification authority (CA). However, exchange of credentials, such as PKCs, is part and parcel of any AGKA and it seems counter-intuitive to be concerned

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 287–308, 2007.

about information leakage. At the same time, in many applications, the identity of the certificate-issuing CA determines the certificate owner's *affiliation*. This is not an issue if affiliation by itself is not a sensitive attribute. However, in certain scenarios, affiliation must be kept private and protected from all unauthorized parties, most commonly, those with different affiliations. We consider two motivating examples.

- CIA agents often operate in hostile environments and their affiliation represents a closely-guarded secret. This is mandated by the rules of the agency. Therefore, if two or more CIA agents need to discover each other and establish a secure communication channel, affiliation-leaking information cannot be exchanged for fear of detection and unpleasant consequences.
- Federal air marshals routinely accompany civilian flights and are required to keep a very low profile, i.e., to blend in as much as possible. When two or more marshals in an airport (or any common vicinity) need to coordinate activities and set up a secure conference, they must do so in an unobservable and undetectable manner, i.e., their affiliations must be kept private.

In a two-party setting, affiliation hiding authentication schemes have been addressed in the past with so-called *secret handshake* protocols [1]. The initial work [1] introduced the notion of privacy in public key-based authentication schemes and proposed the first two-party secret handshake scheme based on bilinear maps and secure under the Gap Diffie-Hellman (GDH) assumption. A subsequent result by Castelluccia, et al. [9] developed a slightly more efficient secret handshake scheme secure under the Computational Diffie-Hellman (CDH) assumption. Both schemes can be used in two versions: If the players use one-time certificates, in addition to affiliation-hiding these protocols trivially attain a property of *unlinkability*, since in addition to not leaking their affiliations, any two instances of the same player cannot be linked with each other. If the players re-use their certificates, the protocols are affiliation-hiding but it's possible to trace multiple occurrences of the same party.

In this paper we consider affiliation hiding in a multi-party (two or more) setting, i.e. for Authenticated *Group* Key Agreement protocols (AGKA). We construct two practical Affiliation-Hiding AGKA protocols (AH-AGKA), wherein participants compute an authenticated common secret key as long as all participants have the same affiliation, i.e., possess certificates issued by the same CA. At the same time, in contrast to a standard AGKA, a party engaging in an AH-AGKA protocol is assured that its affiliation is revealed to only those other protocol participants that belong to the group governed by the same CA. Our protocols have similar properties as the two-party secret handshakes of [1, 9], i.e. they offer affiliation-hiding with standard re-usable certificates, and they can offer unlinkability only if the players use one-time certificates. They can also offer heuristic unlinkability, e.g if players limit the usage of one certificate based on their physical mobility.

Group (or multi-party) secret handshake protocols have been considered in prior work, notably [17] and [12]. In [17], Tsudik and Xu presented the first scheme supporting any number of protocol participants and reusable certificates. However, their approach assures that the participants in the AGKA protocol successfully compute a shared key only if their group revocation information is synchronized (in other words, only if each participant assumes the same revocation *epoch*).

Recently, Jarecki et al. [12] constructed a practical AH-AGKA protocol which avoids this synchronization assumption, based on the (unauthenticated) Burmester-Desmedt group key agreement protocol [7]. However, this AH-AGKA protocol is secure only with the use of one-time certificates.[1] Also, the model of security for AH-AGKA protocols considered in [12] is restricted to a *single instance* of an AH-AGKA protocol execution. Such a model makes sense if each protocol instance uses independent inputs, but it is insufficient in the standard PKI setting of re-usable certificates.

Our Contributions. The contributions of this paper are as follows: First, we upgrade the notion of AH-AGKA in [12] with a more robust and thus more useful notion. The new notion assumes a standard PKI model of re-usable certificates and it is modeled on the standard – and very strong – notion for traditional AGKA protocols in [6, 14], which, in turn, comes from a long line of research on Authenticated 2-party Key Agreement protocols [3, 16, 8]. This upgraded security notion implies that each AH-AGKA protocol session remains secure given arbitrary scheduling of protocol instances and any message-interleaving pattern between these instances, e.g., a man-in-the-middle attack. Also, the security of a protocol session is independent of the usage of keys produced by all *other* protocol sessions.

Second, we construct two AH-AGKA protocols that support standard re-usable certificates and satisfy the new strong notion of AH-AGKA security. Both protocols are implicitly-authenticated variants of the Burmester-Desmedt GKA protocol. These two protocols are secure under the RSA and the GDH assumptions, respectively, in the Random Oracle Model (ROM). (Moreover, the second protocol is secure also under the CDH assumption, but the security reduction from the CDH problem is weaker.) Each scheme involves only 2 communication rounds and few exponentiations per participant. From both communication and computation perspective, the protocol costs are the same as those of the *unauthenticated* Burmester-Desmedt group key agreement protocol [7] and lower than those of the (non affiliation-hiding) signature-based authenticated version of the Burmester-Desmedt protocol due to Katz and Yung [14]. Consequently, our protocols show that *Affiliation Hiding* for AGKA protocol can be achieved at essentially no additional cost. Note, however, that an AH-AGKA protocol guarantees success only if all participants are affiliated with the same CA, which is not the case in a standard AGKA. Moreover, we do not address perfect forward security in this paper.

Third, an independent consequence of our work is a variant of the Burmester-Desmedt GKA protocol which is secure (in ROM), although without perfect forward secrecy, even if the participants re-use their Diffie-Hellman key contributions. The standard Burmester-Desmedt GKA protocol is insecure unless each player uses a new contribution in every protocol instance. As a consequence of re-use of key contributions, this version of the Burmester-Desmedt protocol requires 2 exponentiations per player instead of 3.

Organization. The rest of this paper is organized as follows: Section 2 formally defines an AH-AGKA scheme and the desired security/privacy properties. Section 3 defines the

[1] We want to point out that in addition to requiring more storage for group members, higher load on the issuing CA, and longer certificate revocation structures, a protocol that requires single-use certificates is vulnerable to *depletion attacks*, whereby the adversary repeatedly engages some user in the AH-AGKA protocol, thus depleting the latter's supply of one-time certificates.

cryptographic assumptions required by our constructions. Section 4 presents the RSA-based AH-AGKA protocol, and Section 5 presents the DH-based AH-AGKA protocol. The security proofs for the RSA-based scheme are given in detail, but because of space limitations we relegate the proofs of security for the DH-based scheme to the full version of this paper [13].

2 Affiliation-Hiding Authenticated Group Key Agreement: Model and Definitions

Entities. Our AH-AGKA model is based on the existing standard model for authenticated group key agreement protocols [6, 14]. The main difference is that the standard model assumes a global PKI where each entity has a private/public key-pair and a certificate issued by a CA which is part of the PKI. The PKI involves a *certification hierarchy*, where the integrity of the association between entities and their public keys is vouched by a chain of certificates all leading to some commonly trusted CA-s. In this model, it is assumed that certificates (which in many applications contain information about owners *affiliation*) are publicly available. In contrast, AH-AGKA protocols aim to protect affiliation privacy of the participants and certificates are kept private. Another distinctive feature of our model is its "flat" certification structure, i.e., certification hierarchies and chains are not allowed. There are only CA-s and entities certified by CA-s; there are no intermediate CA-s and no delegation of certificates.

An AH-AGKA *scheme* operates in an environment that includes a set of *users* $\mathcal{U}$ and a set of *groups* $\mathcal{G}$. Each group is administered by a CA responsible for creating the group, admitting entities as members and revoking membership. We assume upper bounds m and l, respectively, on the total number of groups and the number of members in any given group, i.e., $|\mathcal{G}| \leq m$ and $|\mathcal{U}| \leq l$. We assume that each user can be a member of many groups. We denote the fact that user $U \in \mathcal{U}$ is a member of group $G \in \mathcal{G}$ as $U \prec G$. The main part of an AH-AGKA scheme is an AH-AGKA *protocol*, which is executed by any set of users $\Delta = \{U_1, ..., U_n\} \subseteq \mathcal{U}$, for any $n \geq 2$. (More on that below.) Hereafter, the term *group member* refers to a user who is a member of a particular group, whereas the terms *player* and *protocol participant* refer to a user who is currently taking part in some particular instance of an AH-AGKA protocol.

Groups. We note from the outset that the use of the term group is over-loaded in this paper. First, it denotes a set of users with the same affiliation (members of group G), i.e., with certificates issued by the same CA. Second, it refers to an ad-hoc group (Δ) of AH-AGKA protocol participants who may or may not be all members of the same group G. We make the desired meaning unambiguous from the context. We use *protocol participants* or *set of players* when referring to the second meaning, and we use *group* only in the first meaning except when re-using the standard terminology of (Authenticated) *Group* Key Agreement, where the word *Group* refers to the set of players participating in an instance of the AGKE protocol.

AH-AGKA Protocol. Using this terminology, each player $U_i \in \Delta$ participating in an instance of the AH-AGKE protocol executes the protocol instructions on inputs a public key of some group $G \in \mathcal{G}$ s.t. $U_i \prec G$, and U_i's certificate of membership in G. The

purpose of the AH-AGKA protocol is for the players in Δ to establish an authenticated shared secret key as long as (1) each of them run the protocol on the same public key, i.e. the public key of the same group G, and (2) for each $U_i \in \Delta$ it holds that $U_i \prec G$. This key is secret and authenticated in the sense that it can be used for any subsequent secure communication, e.g., entity authentication or message encryption and/or authentication.

To avoid any misunderstanding, we stress that such protocol does not in general imply an efficient solution for an (affiliation-hiding) *group discovery* problem, where each participating player starts a protocol on a *set* of its certificates of membership in a *set* of groups, and the protocol succeeds, for example, as long as all the certificates are valid and all these sets have a non-empty union. In contrast, our AH-AGKA schemes are most practical in scenarios where each user is a member of at most one group. However, we stress that if a user is a member of many groups, this would affect execution efficiency (or robustness), but it would not affect *security* and *affiliation-hiding* of our schemes. Indeed, in the definitions that follow we assume w.l.o.g. that each user is a member of every group.

Public Information and Network Assumptions. In our environment, all groups $G \in \mathcal{G}$ are publicly known. Their CA public keys and certificate revocation lists (CRL-s) maintained by CA-s are publicly accessible. Before any group can be created, a common security parameter must be publicly chosen, and a public Setup procedure is executed on that parameter. The Setup procedure creates common cryptographic parameters which are used as inputs in all subsequent protocols. We stress that the Setup procedure does not need to be executed by a trusted authority: It can be executed by anyone, for example by one of the CAs, and everyone can verify the validity of its outputs.

We assume that communication between users and CA-s, i.e. the certificate issuance process and the CRL retrieval, are conducted over anonymous and authenticated channels. In practice, a user might communicate with the CA, e.g., while retrieving the most recent CRL for its group, over an anonymous channel such as TOR [11]. Alternatively, the CRL-s of all groups can be combined and stored at some highly-available site where they can be either retrieved in bulk (if small) or via some Private Information Retrieval (PIR) protocol, e.g., [10].

We assume that all communication within the AH-AGKA protocol takes place over a broadcast channel. We assume weak synchrony, i.e., the protocol executes in rounds. In practice, this assumption implies that the protocol is started by a broadcast message indicating the number of participants. Weak synchrony among the participants also assumes that the length of the time window assigned to each protocol round is large enough to accommodate clock skews and reasonable communication delays. The broadcast channel is *not* assumed to be authenticated. In fact, the broadcast channel is used for purely notational convenience since we make no assumptions about its reliability. Specifically, when a participant broadcasts a message, it could just as well send a copy of this message to every other participant over a point-to-point link. In our model, the adversary is assumed to have full control of the underlying network: it sees the messages broadcasted by each participant in a given round, and decides which messages will be *delivered* to each participant in that round. The adversary can delete, modify or substitute any message and it can choose to deliver different messages to different participants.

As a consequence of this model, the security and privacy (Affiliation Hiding) properties of our AH-AGKA protocols hold, by definition, given *any* adversarial interference in the protocol. However, we stress that we do not claim any *robustness* properties of our protocols, apart from the basic correctness, i.e. that the protocols succeeds if the players execute the protocols on matching inputs and there is no active adversarial interference in the protocol. Indeed, constructing AH-AGKA protocols which are robust against protocol interference is an open issue.

Player Instances and Protocol Sessions. In line with prior work [8, 6, 14], our model allows for multiple executions of the AH-AGKA protocol scheduled in an arbitrary way, each involving any set of participants. We model this in the usual way, by assuming that every user $U \in \mathcal{U}$ can run multiple *instances* of the protocol. We denote the τ-th instance of user U as Π_U^τ. When player U starts a new instance of the AH-AGKA protocol, it creates a new instance Π_U^τ for a locally unique value τ. Such instances can run on shared state, e.g., certificates and CRLs held by player U, but each instance also keeps separate state. Each player instance can either reject or accept and output a key. We say that an instance Π_U^τ runs a *protocol session*, and we use *player instance* and *protocol session* interchangeably, denoting both as Π_U^τ. When referring to a specific user U_i we use Π_i^τ as a short-hand version of $\Pi_{U_i}^\tau$, to denote τ-th instance of user $U_i \in \mathcal{U}$. Each instance Π_i^τ keeps a state variable, sid_i^τ which can be thought of as a *session id*. (However, see the remark below.) This variable is protocol-dependent, but in our protocols it is always set to an entirety of the communication sent and received by instance Π_i^τ.

AH-AGKA Syntax. We define an AH-AGKA scheme as a collection of the following algorithms:

- Setup: on input of security parameter κ, it generates public parameters params.
- KGen: executed by the group CA, on input params, it outputs the group public key $\mathcal{PK}$ and the corresponding secret key $\mathcal{SK}$ for this group, and an empty certificate revocation list $\mathcal{CRL}$. We denote the group corresponding to the public key $\mathcal{PK}$ as $\mathsf{Group}(\mathcal{PK})$. If $\mathcal{PK}$ was generated by the CA that maintains group G then $\mathsf{Group}(\mathcal{PK}) = G$.
- Add: executed by the CA of group G, on input $\mathcal{SK}$ and $U \in \mathcal{U}$, it adds U to G by generating a certificate for U, denoted cert. If cert is issued under a public key $\mathcal{PK}$, we say that $\mathsf{cert} \in \mathsf{Certs}(\mathcal{PK})$.
- Revoke: executed by the group CA, on input $U \in \mathcal{U}$, it retrieves the corresponding $\mathsf{cert} \in \mathsf{Certs}(PK)$ issued for U, and revokes it by adding a new entry to the group $\mathcal{CRL}$ which uniquely identifies cert. If cert is revoked, we say that $\mathsf{cert} \in \mathsf{RevokedCerts}(\mathcal{CRL})$.
- Handshake: this is the AH-AGKA *protocol* itself, which is an interactive protocol executed by some set of participants $\Delta = \{U_1, ..., U_n\} \subseteq \mathcal{U}$. Each U_i uses its distinct new instance Π_i^τ and runs session Π_i^τ of the protocol on some inputs:

$$(\mathsf{cert}_i^\tau, \mathcal{PK}_i^\tau, \mathcal{CRL}_i^\tau)$$

where $\mathcal{PK}_i^\tau$ is the public key of the group which, in U_i's view, sets the context for the protocol, cert_i^τ is U_i's certificate in $\mathsf{Group}(PK_i^\tau)$, and CRL_i^τ is the CRL of this group.[2] An instance Π_i^τ either rejects or outputs an authenticated secret key K_i^τ.

Remark: Our syntax, though adopted from earlier AGKA models of [6, 14], is slightly different from that used in some other work on Key Agreement protocols, e.g., [8], where a protocol instance takes as additional input, a so-called *session-id* (different from the sid_i^τ value introduced above). In this alternative model, creation of a fresh and locally-unique session-id's, common to all players engaging in the protocol, is assumed to be done before the protocol starts. In contrast, in the model of [6, 14], which we adopt, no such agreed-upon value is assumed. (However, our protocols, similarly to the AGKA protocol in [14], in the first protocol round create a value s which plays the role of such unique session-id input. As a side remark, we point out that unlike the protocol of [14], our AH-AGKA protocol manages to piggyback the creation of this session-id s onto the first round of the protocol, thus saving one communication round.)

Partnering. The purpose of the Handshake protocol is to allow a set of participants with *matching inputs*, i.e. specifying the same group G, to establish a common key. We use the term *session partnering* to denote protocol instances that run on matching inputs and where all protocol messages between them are properly delivered. Namely, we say that a set of protocol instances $\{\Pi_1^{\tau_1}, \Pi_2^{\tau_2}, ..., \Pi_n^{\tau_n}\}$ is *partnered* if there exists a single public key $\mathcal{PK}$ and a single value sid such that, for each session $\Pi_i^{\tau_i}$, in this set it holds that $\mathcal{PK}_i^{\tau_i} = \mathcal{PK}$ and $\mathsf{sid}_i^{\tau_i} = \mathsf{sid}$. The latter implies complete agreement among these player instances with regard to the set of messages sent and delivered between these instances.

Correctness. We say that an AH-AGKA scheme is *correct* if, assuming that all keys, certificates and CRL-s are generated by following the Setup, KGen, Add and Revoke procedures, the following holds:

For any set of *partnered* sessions $\Pi_1^{\tau_1}, \Pi_2^{\tau_2}, ..., \Pi_n^{\tau_n}$ where $\mathsf{cert}_i^{\tau_i} \in \mathsf{Certs}(\mathcal{PK}_i^{\tau_i})$ for each i, and $\mathsf{cert}_i^{\tau_i} \notin \mathsf{RevokedCerts}(CRL_j^{\tau_j})$ for all pairs (i, j), there exists a single unique bit-string K of length κ such that each session $\Pi_i^{\tau_i}$ accepts and outputs $K_i^{\tau_i} = K$.

2.1 Definition of Security

We define AH-AGKA security similarly to standard AGKA protocols in the PKI model, but we must adapt these security notions to our setting. In the setting of an AH-AGKA scheme, the protocol participants, instead of recognizing one another by individual public keys, want to establish authenticated sessions with *any* other participants as long as all these participants are non-revoked members of the same group. This is reflected in the fact that a user starts an AH-AGKE protocol instance on just his certificate and the public key of some chosen group G. One implication this bears for the AH-AGKA

[2] As in standard authentication protocols in the PKI model, the more recent CRL, the better. However, we do not assume that a player has the most recent group CRL.

security definition is that, unlike in a standard AGKA protocol in the PKI model, our notion of security must explicitly include admission and revocation actions of the CA's which manages the groups.

AH-AGKA security is defined via a game between an adversary and a set of users communicating over a network. In this game, the adversary gets to see the public keys of all groups, and some number of certificates in each group, corresponding to all corrupted players and leaked secrets. The adversary then schedules any number of Handshake protocol instances, involving any combination of honest users and groups. The adversary has complete control of the network, i.e., it sees all messages and can delay, delete, modify, or inject any messages received by the honest players. The adversary can also request that some key established in some protocol session be *revealed*. We say that the AGKA protocol is **secure** if, for each (unrevealed) session executed by an honest player, the adversary cannot distinguish the key output by the player on that session from a random bitstring of the same length. (As discussed below, the only exception is if the adversary previously requested that a key be revealed for some protocol session *partnered* with the one at hand.)

Formally, security is defined via an interaction of an adversarial algorithm $\mathcal{A}$ and a challenger $\mathcal{C}$ on common inputs (κ, l, m). The interaction starts with $\mathcal{C}$ generating params via $\mathsf{Setup}(\kappa)$, and initializing m groups $G_1, ..., G_m$, by running the $\mathsf{KGen}(\mathsf{params})$ algorithm m times. $\mathcal{C}$ initializes all members in these groups, by running the $\mathsf{Add}(\mathcal{SK}_j)$ algorithm, for each $\mathcal{SK}_j$, $j = 1, ..., m$, for l times. This way, $\mathcal{C}$ generates m certificates for every $U \in \mathcal{U}$, thus making every user a member of every group. The adversary $\mathcal{A}$ gets all generated public keys $\mathcal{PK}_1, ..., \mathcal{PK}_m$. It then chooses any subset $\mathsf{Rev} \subseteq \mathcal{U}$ of initially corrupted players and gets the set of their certificates $\{\mathsf{cert}_i{}^{(j)} \mid U_i \in \mathsf{Rev}, j \in \{1, ..., m\}\}$. For each group G in $\mathcal{G}$, the challenger runs the Revoke algorithm to revoke all corrupted members $U \in \mathsf{Rev}$, and outputs the resulting CRL-s for each group, i.e., $\mathcal{CRL}_1, ..., \mathcal{CRL}_m$.

After this initialization, $\mathcal{A}$ schedules any number of Handshake protocols, arbitrarily manipulates their messages, requests the keys on any number of the (accepting) sessions, and optionally corrupts any number of additional players, all of which can be modeled by $\mathcal{A}$ issuing any number of the commands listed below. Finally, $\mathcal{A}$ stops and outputs a single bit b'. The commands the adversary can issue, and the way the challenger $\mathcal{C}$ responds to them, are listed below. In all commands we assume that $U \in \mathcal{U} \setminus \mathsf{Rev}$.

- $\mathsf{Start}(U, G)$: If $U = U_i$ for some $U_i \in \mathcal{U} \setminus \mathsf{Rev}$ and $G = G_j$ for some $G_j \in \mathcal{G}$, the challenger retrieves key $\mathcal{PK}_j$ for group G_j, certificate $cert_i{}^{(j)}$ issued to player U_i for group G_j, and the $\mathcal{CRL}_j$ for group G_j, and initiates instance Π_U^τ, where τ is an index that has not been used by user U before. The challenger follows the Handshake protocol on behalf of instance Π_U^τ on inputs $(\mathsf{cert}_i{}^{(j)}, \mathcal{PK}_j, \mathcal{CRL}_j)$, forwarding any message generated by this instance to $\mathcal{A}$. The challenger keeps the state of all initiated instances. We denote the group upon which Π_U^τ is initiated as $\mathsf{Group}(\Pi_U^\tau)$. If Π_U^τ is triggered on G_j then $\mathsf{Group}(\Pi_U^\tau) = G_j$. $\mathcal{C}$ also hands to $\mathcal{A}$ the index τ of this instance.
- $\mathsf{Send}(U, \tau, \mathcal{M})$: If instance Π_U^τ has been initiated and is still active, $\mathcal{C}$ delivers a *set* $\mathcal{M}$ of messages to this instance. The set $\mathcal{M}$ should normally contain $n - 1$

messages $M_2, ...M_n$, for $n \geq 2$, which models the messages that instance Π_U^τ receives in the current round of this protocol. The instance interprets these messages as broadcasted by $n-1$ distinct instances of the protocol in the same round. ($\mathcal{A}$ could send an empty set $\mathcal{M}$, but an instance would invariable immediately abandon the protocol as a result.) $\mathcal{C}$ forwards to $\mathcal{A}$ any message Π_U^τ generates in response. If Π_U^τ outputs a key, $\mathcal{C}$ stores this key with the session state.

- $\mathsf{Reveal}(U, \tau)$: If Π_U^τ outputs a session key K, $\mathcal{C}$ sends K to $\mathcal{A}$. If the session has either not completed yet or has been rejected, $\mathcal{C}$ sends $\mathcal{A}$ a null value.
- $\mathsf{Test}(U, \tau)$: This query is allowed only once. If session Π_U^τ has output a session key K, $\mathcal{C}$ picks a random bit b. If $b = 1$, then $\mathcal{C}$ sends K to $\mathcal{A}$. If $b = 0$ then $\mathcal{C}$ sends to $\mathcal{A}$ a random κ-bit long value K', instead of K. If the session does not exist, failed, or is still active, the challenger ignores this command.

Session Freshness and Legitimate Adversaries. We call an active session Π_U^τ of an uncorrupted player U *fresh*, if for all sessions $\Pi_{U'}^{\tau'}$ partnered with Π_U^τ the adversary has not queried $\mathsf{Reveal}(U, \tau)$ or $\mathsf{Reveal}(U', \tau')$. Note that the adversary knows whether any two sessions are partnered or not. We call an adversary $\mathcal{A}$ *legitimate* if it poses a Test query on a fresh session Π_i^τ, and afterwards $\mathcal{A}$ does not issue a Reveal query on Π_U^τ or any $\Pi_{U'}^{\tau'}$ partnered with Π_U^τ.

Definition 1. *Denote the final output of adversary $\mathcal{A}$ in the above interaction with the challenger $\mathcal{C}$ on common inputs (κ, l, m) as $\langle \mathcal{A}, \mathcal{C}(b) \rangle (\kappa, l, m)$. We define the adversary's advantage in the security game as*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathrm{sec}} = |\Pr[b = b' \mid b' \leftarrow \langle \mathcal{A}, \mathcal{C}(b) \rangle (\kappa, l, m)] - 1/2|$$

where the probability is taken over the random coins used by $\mathcal{A}$ and $\mathcal{C}$ and a random choice of the challengers bit b.

We call an AH-AGKA scheme $(\epsilon, t, q_s, q_H, l, m)$-secure *in the Random Oracle Model if for all legitimate adversaries $\mathcal{A}$ who run in time t, start q_s AH-AGKA sessions, and make q_H hash function queries, it holds that $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{sec}} \leq \epsilon$.*

2.2 Definition of Affiliation-Hiding

We define the affiliation-hiding property using a similar game as in the security definition in the previous section. However, the adversary's goal in the affiliation-hiding game is not to violate semantic security of some session key (as in the security game above) but to learn the participants' affiliation. We model the property of the attacker's *inability* to learn the affiliation by comparing two executions of the adversary: one where the challenger follows the protocols faithfully on behalf of all honest participants, and the other where the adversary interacts with a *simulator*, instead of the real users. The simulator attempts to follow the adversary's instructions, except that it is never told the groups for which the (scheduled by the adversary) $\mathsf{Handshake}$ protocol instances are executed, i.e., if the adversary issues a $\mathsf{Start}(U, G)$ query, the simulator gets only an identifier $(\hat{id})$ which is uniquely but arbitrarily assigned to the pair $(U, G) \in \mathcal{U} \times \mathcal{G}$.

Consequently, these inputs are also the only thing that the adversary can possibly learn from the interaction with a simulator. The simulated protocol messages can reveal only whether or not two sessions involve *the same* (user,group) pair. However, the

adversary does not learn which group, nor can he decide if two instances of two different users belong to the same group. Note that we allow the adversary to be able to *link* instances which involve the same (user,group) pair because the simulator gets the same $\hat{id}$ for such instances. Indeed, all AH-AGKA schemes we propose in this paper are linkable in this sense.

Formally, the game between $\mathcal{A}$ and $\mathcal{C}^{\mathsf{ah}}$, on common inputs κ, l, m, starts exactly as the game between $\mathcal{A}$ and $\mathcal{C}$ in the security definition above. Namely, $\mathcal{C}^{\mathsf{ah}}$ runs $\mathsf{Setup}(\kappa) \rightarrow \mathsf{params}$, then runs m instances of $\mathsf{KGen}(\mathsf{params}) \rightarrow (\mathcal{PK}_j, \mathcal{SK}_j)$, for $j = 1, .., m$, then lm instances of the Add algorithm, $\mathsf{Add}(\mathcal{SK}_j) \rightarrow \mathsf{cert}_i^{(j)}$, for $i = 1, ..., l$ and $j = 1, ..., m$, which generate l certificates for each of the m groups. $\mathcal{C}^{\mathsf{ah}}$ gives to $\mathcal{A}$ all public keys $\mathcal{PK}_j$ and the certificates of all corrupted users: $\{\mathsf{cert}_i^{(j)} \mid i \in \mathsf{Rev}, j \in \{1, ..., m\}\}$, revokes all of these certificates, and finally publishes the resulting CRL-s.

After this initialization, $\mathcal{A}$ schedules any number of Handshake instances and manipulates their messages in arbitrary ways. We model this interaction between $\mathcal{A}$ and $\mathcal{C}^{\mathsf{ah}}$ by allowing $\mathcal{A}$ any number of queries $\mathsf{Start}(U, G)$ and $\mathsf{Send}(U, \tau, \mathcal{M})$ to $\mathcal{C}$, as in the security game.

However, $\mathcal{A}$ does not make a Test query in this game. Instead, $\mathcal{C}^{\mathsf{ah}}$ picks a random bit b at the beginning of the execution and performs $\mathcal{A}$'s commands depending on the value of b. If $b = 0$, $\mathcal{C}^{\mathsf{ah}}$ responds to $\mathcal{A}$'s commands $\mathsf{Start}(U, G)$ and $\mathsf{Send}(U, s, \mathcal{M})$ by following the corresponding protocol on behalf of the user, exactly as in the security game in above. Otherwise ($b = 1$), $\mathcal{C}^{\mathsf{ah}}$ replies to $\mathcal{A}$ with messages produced by the simulator $\mathcal{SIM}$, which is an interactive machine which runs only on inputs params, and, instead of $\mathsf{Start}(U, G)$ and $\mathsf{Send}(U, s, \mathcal{M})$, it gets on-line inputs $\mathsf{Start}(\hat{id})$ and $\mathsf{Send}(\hat{id}, \mathcal{M})$, respectively, where $\hat{id}$ is a unique (and random) string assigned to this (U, G) pair. At the end of the game, the adversary outputs a bit b'.

Definition 2. *Denote the final output of adversary $\mathcal{A}$ in the above interaction with the challenger $\mathcal{C}^{\mathsf{ah}}$ on common inputs (κ, l, m) as $\langle \mathcal{A}, \mathcal{C}^{\mathsf{ah}}(b)\rangle(\kappa, l, m)$. We define the adversarial advantage in the affiliation-hiding game as*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ah}}(\kappa, l, m) = |\Pr[b = b' \mid b' \leftarrow \langle \mathcal{A}, \mathcal{C}^{\mathsf{ah}}(b)\rangle(\kappa, l, m)] - 1/2|$$

where the probability is taken over the random coins used by $\mathcal{A}$ and $\mathcal{C}^{\mathsf{ah}}$ and a random choice of the challengers bit b.

We call an AH-AGKA scheme affiliation-hiding *if for any probabilistic polynomial-time adversary $\mathcal{A}$, for parameters l an m polynomially related to κ, the adversarial advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ah}}(\kappa, l, m)$ is a negligible function of κ.*

Remark on the Affiliation-Hiding Notion. First, note that the above definition restricts $\mathcal{A}$ to only Start and Send queries. This results in a restricted notion of affiliation-hiding, which can and should be strengthened to include the information the $\mathcal{A}$ can gain about session keys from higher-level protocols, modeled by Reveal queries. Such strengthening is in fact necessary in practice because without the Reveal queries, $\mathcal{A}$'s view does not even contain information on whether a given session instance failed or succeeded, which is something that a network adversary can very often learn in practice. We leave consideration of stronger notions of affiliation-hiding to the full version of the paper. Second, an exact-security version of the above notion can be easily extrapolated,

and this too will be included in the full version, together with the exact security bounds on affiliation-hiding for the two AH-AGKA schemes presented in this paper.

3 Cryptographic Assumptions

Definition 3. *Let* $\mathsf{S\text{-}RSA\text{-}IG}(\kappa)$ *be an algorithm that outputs so-called safe RSA instances, i.e. pairs* (n, e) *where* $n = pq$*,* e *is a small prime that satisfies* $gcd(e, \phi(n)) = 1$*, and* p, q *are randomly generated* κ*-bit primes subject to the constraint that* $p = 2p' + 1$*,* $q = 2q' + 1$ *for prime* p', q'*,* $p' \neq q'$*.*

We say that the RSA problem is (ϵ, t)*-hard on* 2κ*-bit safe RSA moduli, if for every algorithm* $\mathcal{A}$ *that runs in time* t *we have*

$$\Pr[(n, e) \leftarrow \mathsf{S\text{-}RSA\text{-}IG}(\kappa), g \leftarrow \mathbb{Z}_n^* : \mathcal{A}(n, e, g) = z \text{ s.t. } z^e = g \pmod{n}] \leq \epsilon.$$

Definition 4. *Let* $\mathbb{G}$ *be a cyclic group of prime order* q *with a generator* g*. We say that the Square Diffie-Hellman Problem (SDH) in* $\mathbb{G}$ *is* (ϵ, t)*-hard if for every algorithm* A *running in time* t *we have*

$$\Pr[x \leftarrow \mathbb{Z}_q : A(g, g^x) = g^{x^2}] \leq \epsilon.$$

DDH oracle: A DDH oracle in group $\mathbb{G}$ is an algorithm that returns 1 on queries of the form (g, g^x, g^y, g^z) where $z = xy \bmod q$, and 0 on queries of the form (g, g^x, g^y, g^z) where $z \neq xy \bmod q$.

Definition 5. *We say that the Gap Square Diffie-Hellman Problem (GSDH) in group* $\mathbb{G}$ *is* (ϵ, t)*-hard if for every algorithm* A *running in time* t *on access to the DDH oracle* $\mathsf{DDH}_{\mathbb{G}}$ *in group* $\mathbb{G}$ *we have*

$$\Pr[x \leftarrow \mathbb{Z}_q : A^{\mathsf{DDH}_{\mathbb{G}}}(g, g^x) = g^{x^2}] \leq \epsilon.$$

It is well known that the SDH problem is equivalent to the computational Diffie-Hellman (DH) problem. Just note that $g^{xy} = (g^{(x+y)^2}/(g^{x^2} g^{y^2}))^{2^{-1}}$, and that oracle errors can be easily corrected since both the SDH and the DH problems are random self-reducible. Similarly, the GSDH problem is equivalent to the Gap Diffie-Hellman problem (GDH), which is believed to be hard in many prime-order groups. In particular, generic group algorithms cannot solve it in time better than $\Omega(\sqrt{q})$ [5].

4 AH-AGKE Scheme Based on the RSA Assumption

- Setup: On security parameter κ, the Setup procedure picks two other parameters κ', and κ''. Parameter κ is the length of the key output by the key agreement protocol Handshake, κ' is an additional parameter which in practice can be 160, and κ'' is chosen so that the RSA problem for $2\kappa''$-bit safe RSA moduli has at least κ-bit security (see theorem 1 for exact bounds). Whenever we say that two distributions D_1, D_2 are statistically close we mean that the statistical difference between them is bounded by $O(2^{-min(\kappa, \kappa', \kappa'')})$. The setup procedure also chooses a κ'-bit prime $\hat{q}$ and defines hash functions $H_{\hat{q}} : \{0,1\}^* \rightarrow Z_{\hat{q}}^*$ and $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$.

- KGen: Generate a $2\kappa''$-bit safe RSA modulus $n = pq$, where $p = 2p'+1$, $q = 2q'+1$, and p, q, p', q' are primes. Pick a random element g s.t. g generates a maximum subgroup in Z_n^*, i.e. $ord(g) = 2p'q'$, and s.t. $-1 \notin \langle g \rangle$. (This holds for about half of the elements in Z_n^*, and it is easily tested.) Note that in this case $Z_n^* \equiv \langle -1 \rangle \times \langle g \rangle$. Therefore, in particular, if $x \leftarrow Z_{2p'q'}$ and $b \leftarrow \{0,1\}$ then $(-1)^b g^x$ is distributed uniformly in Z_n^*. RSA exponents (e, d) are chosen in the standard way, as a small prime e and $d = e^{-1} \pmod{\phi(n)}$. The secret key is (p, q, d) and public key is (n, g, e). Key generation also fixes a hash function $H_n : \{0,1\}^* \rightarrow Z_n$, specific to the group modulus n.[3]
- Add: To add user U to the group, the manager picks a random string $id \leftarrow \{0,1\}^{\kappa'}$ and computes a (full-domain hash) RSA signature on id, $\sigma = h^d \pmod{n}$, where $h = H_n(id)$. U's certificate is $\mathsf{cert} = (id, \sigma)$.
- Revoke: To remove user U from the group, the manager appends string id to the group $\mathcal{CRL}$, where (σ, id) is U's certificate in this group.
- Handshake: This is an AGKA protocol executed by some set $\Delta = \{U_1, ..., U_n\}$ of players. Each player U_i starts a session $\Pi_i^{\tau_i}$ for a (locally) fresh τ_i, on *some* inputs $(\mathsf{cert}_i, (n, e, g), \mathcal{CRL}_i)$ s.t. (n, e, g) is some public key, $\mathsf{cert}_i = (id_i, \sigma_i)$ is U_i's certificate for this public key (n, e, g), i.e. $\mathsf{cert}_i \in \mathsf{Certs}(n, e, g)$, and $\mathcal{CRL}_i$ is the (hopefully recent) CRL for group $\mathsf{Group}(n, e, g)$. The Handshake protocol is in Figure 1 below (see also the note below).

Notational Simplifications. In figure 1, we make several assumptions to simplify the notation. First, we denote the set of participating players as simply $U_1, ..., U_n$, even though they can be any n users $U_{i_1}, ..., U_{i_n}$ among $\mathcal{U} = \{U_1, ..., U_l\}$, for any $n \geq 2$. Secondly, we assume that the order between the players, which in the full protocol is determined on-line according to the players' messages in Round 1, is simply $U_1, ..., U_n$.[4] For simplicity of notation, we assume that the indices cycle modulo n, i.e. $U_{n+1} = U_1$. We also assume that each instance Π_i^{τ} starts on the same public key (n, e, g). (Since we are not concerned with robustness properties in this paper, we do not concern ourselves with what happens with executions of instances which are not *partnered*, and in particular do not run on the same public keys.)

Affiliation hiding property of the protocol in figure 1 depends crucially on the fact that if the distribution of variable $\bar{\theta}_i$ is indistinguishable from uniform over Z_n then the distribution of $\theta_i = \bar{\theta}_i + kn$ is statistically close to $U_{2^{2\kappa''+\kappa}}$. There is an alternative way

[3] Selecting separate hash function H_n for every group is done purely for notational convenience. A family of hash functions $H_n : \{0,1\}^* \rightarrow Z_n$ s.t. each H_n is statistically close to a random function with range Z_n, can be easily implemented in the random oracle model with a single hash function with range $2^{2\kappa''+\kappa}$. E.g., $H_n(m) = H(n, m) \bmod n$.

[4] This ordering is done as follows: In the protocol each player U_i picks a long-enough random nonce μ_i (see Round 1 in figure 2), which U_i then includes in *all* its messages in the protocol. After receiving some set of messages in Round 1 (note that we assume weak synchrony), every receiver sorts all the received messages by the increasing order of these μ_i values. Each player then (re)labels all the participants and the messages received in Round 1, including its own messages and its inputs, according to this order. The actual protocol runs exactly as the simplified protocol in figure 2 in the case that this ordering of players created in Round 1 coincides with the original labels $i = 1, ..., n$ of the participants $U_1, ..., U_n$ assumed for simplicity in figure 2.

The inputs of instance Π_i^τ of player U_i are $\mathsf{cert}_i = (\sigma_i, id_i)$, (n, g, e), and $\mathcal{CRL}_i$. Note that $\sigma_i^e = H_n(id_i) \bmod n$.

[Round 1]: U_i picks random values $b_i \leftarrow \{0,1\}$, $t_i \leftarrow Z_{n/2}$, and $\mu_i \leftarrow \{0,1\}^{3\kappa}$, computes $\bar{\theta}_i = (-1)^{b_i}\sigma_i g^{t_i} \pmod n$, sets $\theta_i = \bar{\theta}_i + \nu n$ for random $\nu \leftarrow [0, ..., \lfloor 2^{2\kappa''+\kappa}/n \rfloor]$, and U_i broadcasts (θ_i, id_i, μ_i).

- Assume that player U_i received n-1 messages (θ_1, id_1, μ_1) , ..., $(\theta_{i-1}, id_{i-1}, \mu_{i-1})$, $(\theta_{i+1}, id_{i+1}, \mu_{i+1})$, (θ_n, id_n, μ_n) in Round 1. (This is a simplification. In the real protocol each receiver U_i orders the received messages, and the players which sent them, according to values μ these messages contain. See footnote 4.)
  If any two messages contain the same value id_j or the same value μ_j, player U_i rejects.
- U_i sets $s = ((n, g, e), \{\theta_j, id_j, \mu_j\}_{j=1,...,n})$
- If $id_j \in \mathcal{CRL}_i$ for any j then U_i picks a random value X_i in $Z_{\hat{q}}^*$ and sets $\mathsf{reject} = T$. Otherwise, U_i computes $X_i = H_{\hat{q}}((z_{i+1})^{t_i}, s)/H_{\hat{q}}((z_{i-1})^{t_i}, s) \pmod{\hat{q}}$, where $z_{i+1} = (\theta_{i+1})^{2e}(h_{i+1})^{-2} \pmod n$ and $z_{i-1} = (\theta_{i-1})^{2e}(h_{i-1})^{-2} \pmod n$.
  (Note that if (id_j, σ_j) is a certificate for public key (n, e, g) and $\theta_j = (-1)^{b_j}\sigma_j g^{t_j} + \nu n$ then $z_j = g^{2et_j}$.)

[Round 2]:

U_i broadcasts (X_i, μ_i).

- If in Round 2 player U_i receives n-1 values X_j accompanied by μ_j's that match the $\mu_1, ..., \mu_{i-1}, \mu_{i+1}, ..., \mu_n$ values above, if $\prod_{j=1}^n X_j = 1$, and if $\mathsf{reject} \neq T$, then U_i computes $k_i = H_{\hat{q}}((z_{i-1})^{t_i}, s)^n \cdot (X_i)^{n-1} \cdot (X_{i+1})^{n-2} \cdots X_{i-2} \pmod{\hat{q}}$ and outputs $K_i = H(k_i, \mathsf{sid}_i)$, where $\mathsf{sid}_i = ((n, g, e), \{\theta_j, id_j, \mu_j, X_j\}_{j=1,...,n})$. Otherwise U_i rejects.

Fig. 1. RSA-based Affiliation-Hiding AGKE protocol

that we can use to hide the range of θ_i, which does not take the κ bandwidth overhead [2], which is to repeat picking $\bar{\theta}_i$ until $\bar{\theta}_i \in \{0,1\}^{2\kappa''-1}$. However, the expected running time of such procedure is twice larger than ours. Moreover, such procedure can be subject to timing attacks. Note that the overhead of κ bits our procedure incurs is small compared to $|\bar{\theta}_i| = |n|$.

Protocol Correctness. To see that the protocol Handshake in Figure 1 is correct, note that if some n sessions $\Pi_1^{\tau_1}, ..., \Pi_n^{\tau_n}$ are partnered then they all run on the same public key (n, e, g), and all the values $(\theta_1, id_1, \mu_1, X_1)$,...,$(\theta_n, id_n, \mu_n, X_n)$ are exchanged between them without interference. Therefore, first of all, each participating player will create the same order among the participants, and hence each player labels all the exchanged values in the same way, so we can assume for simplicity that this ordering coincides with the original labels $i = 1, ..., n$. Each player also computes also the same value s and sid_i. To see that each player computes the same value k_i and hence the same key K_i, note that for each j we have $z_j = \theta_j^{2e} h_j^{-2} = g^{2et_j}$, and therefore, each $X_i = H_{\hat{q}}(g^{2et_it_{i+1}}, s)/H_{\hat{q}}(g^{2et_{i-1}t_i}, s) \pmod{\hat{q}}$. Note also that $H_{\hat{q}}((z_{i-1})^{t_i}, s) = H_{\hat{q}}(g^{2et_{i-1}t_i}, s)$. It follows that for every i we have

$$k_i = H_{\hat{q}}(g^{2et_{i-1}t_i}, s) * H_{\hat{q}}(g^{2et_it_{i+1}}, s) * ... * H_{\hat{q}}(g^{2et_{i-2}t_{i-1}}, s) \bmod \hat{q}$$

Therefore all the keys K_i are the same as well.

Theorem 1. *Assuming that the RSA problem is (ϵ', t')-hard on random safe RSA moduli of length $2\kappa''$, the above tuple of algorithms* (Setup, KGen, Add, Revoke, Handshake) *is an $(\epsilon, t, q_s, q_H, l, m)$-secure AH-AGKE scheme in the Random Oracle Model as long as*

$$\epsilon \approx m * (2\epsilon' + 2lq_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_s 2^{-\kappa''+2})$$
$$t \approx t' - (m * t_{kg} + q_s * q_H * t_{exp})$$

where t_{kg} is the time to generate an RSA private/public key pair and t_{exp} is the time of (multi)exponentiation modulo n, for $2\kappa''$-bit RSA moduli.

Proof. Assume a legitimate PPT adversary $\mathcal{A}$ interacting with challenger $\mathcal{C}$ as described in the security definition (definition 1). Assume that there are m groups and l users in the universe, and that $\mathcal{A}$ runs in time t, starts at most q_s sessions, and makes at most q_H queries to the hash functions H_n, $H_{\hat{q}}$, and H. Assume w.l.o.g. that $\mathcal{A}$ always makes a test query on some session. Denote $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sec}} = |Pr[b' = b]|$, i.e. the advantage of the adversary $\mathcal{A}$ in the interaction with $\mathcal{C}$, by ϵ. We split the security proof into two parts. First we describe the simulation procedure, $\mathcal{SIM}$, which using $\mathcal{A}$, attempts to solve for z s.t. $z^e = g \bmod n$ on an RSA challenge (n, e, g). This simulation procedure will run in time t' approximately $t + (m * t_{kg} + q_s * q_H * t_{exp})$. We will then argue that the probability of $\mathcal{SIM}$'s success in solving the RSA challenge is at least $\epsilon' \geq \epsilon/m - (2lq_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_s 2^{-\kappa''+2})$, assuming that element g in $\mathcal{SIM}$'s challenge is such that $\langle -1 \rangle \times \langle g \rangle = Z_n^*$. Note that if n is a safe RSA modulus then for a random $g \in Z_n^*$ this holds with probability $1/2 - O(2^{-|n|/2}) \approx 1/2$. Therefore, the success of $\mathcal{SIM}$ on solving a random $g \in Z_n^*$ is (statistically close to) at least half the above expression, which completes the proof.

PART I: CONSTRUCTION OF A SIMULATOR

Setup. Given the RSA challenge (n, e, g), $\mathcal{SIM}$ follows the Setup algorithm with parameters κ, κ', and $\kappa'' = |n|/2$. As mentioned above, we assume that $\langle -1 \rangle \times \langle g \rangle = Z_n^*$.

Initialization of all groups. Let $G^* \in \mathcal{G}$ be a group s.t. the probability that the adversary $\mathcal{A}$ tests on Π_i^τ s.t. $\mathsf{Group}(\Pi_i^\tau) = G^*$ is not less than $1/m$. (Recall that we assume $\mathcal{A}$ always tests some session.) Simulator $\mathcal{SIM}$ initializes all the groups in $\mathcal{G}$ except G^* as in the real protocol. $\mathcal{SIM}$ also creates l certificates for each of these groups by following the Add procedure, and in the rest of the simulation $\mathcal{SIM}$ simply follows the Handshake protocol on behalf of all instances Π_i^τ s.t. $\mathsf{Group}(\Pi_i^\tau) \neq G^*$. Thus, in the rest of the simulation description we will only describe $\mathcal{SIM}$'s actions with regard to instances Π_i^τ s.t. $\mathsf{Group}(\Pi_i^\tau) = G^*$.

For group $\mathcal{G}^*$, $\mathcal{SIM}$ sets its public key as (n, e, g), and creates the certificates for each revoked player $U_i \in \mathsf{Rev}$ by simulating an RSA signature (id_i, σ_i) under key (n, e, g). Namely, $\mathcal{SIM}$ picks two random values $id_i \leftarrow \{0,1\}^{\kappa'}$ and $\sigma_i \leftarrow Z_n^*$, and assigns $H_n(id_i)$ to $\sigma_i^e \pmod n$. If $\mathcal{A}$ has already queried H_n on any id_i's chosen by $\mathcal{SIM}$ in this way, $\mathcal{SIM}$ abandons the simulation. For each $U_i \notin \mathsf{Rev}$ in G^*, $\mathcal{SIM}$ picks a random value $id_i \leftarrow \{0,1\}^{\kappa'}$. $\mathcal{SIM}$ hands to $\mathcal{A}$ all the public keys and the certs of the corrupted players.

Hash queries to H_n, $H_{\hat{q}}$ and H. For each query x to H_n, $\mathcal{SIM}$ picks random $a \leftarrow Z_n^*$ and sets $H_n(x) = a^e \cdot g^{-1} \pmod{n}$. W.l.o.g, assume that H_n is queried on each id_i for $U_i \notin \mathsf{Rev}$. Denote value a chosen above for $x = id_i$ as a_i, and $H_n(id_i) = a_i^e g^{-1}$ as h_i. For the queries to H and $H_{\hat{q}}$, $\mathcal{SIM}$ simply passes these queries to H and $H_{\hat{q}}$, respectively. However, for each query (r, s) to $H_{\hat{q}}$, $\mathcal{SIM}$ also tries to solve the RSA challenge as we describe below.

After the above initialization, $\mathcal{SIM}$ must provide responses for $\mathcal{A}$'s queries Start, Send, Reveal, and Test, which would look to $\mathcal{A}$ as the real execution, i.e. as in $\mathcal{A}$'s interaction with challenger $\mathcal{C}$. For notational convenience assume that the local index τ of each instance Π_i^τ is globally unique (e.g., assume that τ in Π_i^τ has a suffix i). In the following description, we add as a superscript the instance index τ to all values related to Π_i^τ. For example, θ_i^τ, X_i^τ will refer to messages θ_i, X_i sent by instance Π_i^τ. As mentioned above, $\mathcal{SIM}$ responds to $\mathcal{A}$'s commands relating to instances Π_i^τ s.t. $\mathsf{Group}(\Pi_i^\tau) \neq G^*$ by simply following the honest players' protocol. However, for queries involving group G^*, simulator $\mathcal{SIM}$ respondes as follows:

Start commands. For the $\mathsf{Start}(U_i, G^*)$ command, $\mathcal{SIM}$ initializes instance Π_i^τ. $\mathcal{SIM}$ picks $b_i^\tau \leftarrow \{0,1\}$, $\gamma_i^\tau \leftarrow Z_{n/2}$, and computes $\bar{\theta}_i^\tau = (-1)^{b_i^\tau} \cdot a_i \cdot g^{\gamma_i^\tau} \pmod{n}$. Notice that the distribution of $\bar{\theta}_i^\tau$ in this simulation and in the real protocol are statistically close because both are statistically close to uniform in Z_n^*. Note that since $h_i = (a_i)^e/g$, therefore $\bar{\theta}_i^\tau = (-1)^{b_i^\tau} \cdot (h_i g)^d \cdot g^{\gamma_i^\tau} = (-1)^{b_i^\tau} \cdot h_i^d \cdot g^{d+\gamma_i^\tau} = (-1)^{b_i^\tau} \cdot h_i^d \cdot g^{t_i^\tau} \pmod{n}$, where $t_i^\tau = \gamma_i^\tau + d \pmod{\phi(n)/2}$. The simulator does not know either d or t_i^τ, but will use the above relation to solve for g^d later. $\mathcal{SIM}$ also chooses $\mu_i^\tau \leftarrow \{0,1\}^{3\kappa}$, $\nu_i^\tau \leftarrow [0, ..., \lfloor 2^{2\kappa''+\kappa}/n \rfloor]$, sets $\theta_i^\tau = \bar{\theta}_i^\tau + \nu_i^\tau$ and replies with $(\theta_i^\tau, id_i, \mu_i^\tau)$.

Send queries. Consider an instance Π_i^τ created by the Start command above. We denote the Send command to this instance corresponding to Round 1 of the protocol by Send_1, and the Send command corresponding to Round 2 of the protocol by Send_2. In the following statement, just like we did in the description of the protocol, we assume that the index of player U_i involved in session Π_i^τ belongs to set $i \in \{1, .., n\}$. (In general $i \in \{1, .., l\}$ where $l = |\mathcal{U}| > n$, but the proof in the general case is easy to extrapolate from the proof we give here.)

For the $\mathsf{Send}_1(U_i, \tau, \{\hat{\theta}_j^\tau, \hat{id}_j^\tau, \hat{\mu}_j^\tau\}_{j=1,..,n,j\neq i})$ command, unless there are collisions in id's or μ's, $\mathcal{SIM}$ sets s_i^τ as in the protocol. If any $\hat{id}_j^\tau$'s are on $\mathcal{CRL}_i$ then $\mathcal{SIM}$ sets $X_i^\tau \leftarrow Z_{\hat{q}}^*$ and $\mathsf{reject}_i^\tau = T$. Otherwise, $\mathcal{SIM}$ sets $X_i^\tau = c_{i,i+1}^\tau / c_{i,i-1}^\tau \pmod{\hat{q}}$ where values $c_{i,j}^\tau$ for $j = i-1$ and $j = i+1$ are chosen as follows. If $\exists$ some $\Pi_{i'}^{\tau'}$ which received the Send_1 query s.t.:

1. $(\theta_{i'}^{\tau'}, id_{i'}^{\tau'}, \mu_{i'}^{\tau'}) = (\hat{\theta}_j^\tau, \hat{id}_j^\tau, \hat{\mu}_j^\tau)$
2. $(\hat{\theta}_{j'}^{\tau'}, \hat{id}_{j'}^{\tau'}, \hat{\mu}_{j'}^{\tau'}) = (\theta_i^\tau, id_i^\tau, \mu_i^\tau)$ for $j' = i' + 1$ or $j' = i' - 1$
3. $s_{i'}^{\tau'} = s_i^\tau$ and $\mathsf{reject}_{i'}^{\tau'} \neq T$

then $\mathcal{SIM}$ assigns $c_{i,j}^\tau \leftarrow c_{i',j'}^{\tau'}$, where $c_{i',j'}^{\tau'}$ is a value $\mathcal{SIM}$ has previously chosen when dealing with the Send_1 command to session $\Pi_{i'}^{\tau'}$. Note that this case corresponds to an adversary who honestly routes the messages of matching instances Π_i^τ and $\Pi_{i'}^{\tau'}$

from one to another. In such case in the real execution these two instances would compute the same value $c^\tau_{i,j} = c^\tau_{i',j'}$, where

$$c^\tau_{i,j} = H_{\hat{q}}((\hat{z}^\tau_j)^{t^\tau_i}, s^\tau_i) \quad \text{and} \quad c^{\tau'}_{i',j'} = H_{\hat{q}}((\hat{z}^{\tau'}_{j'})^{t^{\tau'}_{i'}}, s^{\tau'}_{i'})$$

If any of these conditions are not met, which corresponds to the case where there is no instance $\Pi^{\tau'}_{i'}$ which runs on matching inputs as Π^τ_i, or when the adversary actively interferes in the communication between these two instances, $\mathcal{SIM}$ picks a fresh random value $c^\tau_{i,j} \leftarrow Z^*_{\hat{q}}$. In both cases $\mathcal{SIM}$ stores $[j, \Pi^\tau_i, s^\tau_i, (\hat{\theta}^\tau_j, \hat{id}^\tau_j, \hat{\mu}^\tau_j), c^\tau_{i,j}]$ in a table denoted $T_{H_{\hat{q}}}$. Finally, $\mathcal{SIM}$ replies with (X^τ_i, μ^τ_i).

For all the $\mathsf{Send}_2(U_i, \tau, \{\hat{X}^\tau_j, \hat{\mu}^\tau_j\}_{j=1,..,n,j\neq i})$ commands, $\mathcal{SIM}$ abandons Π^τ_U if values $\hat{\mu}^\tau_j$ are not correct or $\Pi^n_{j=1}\hat{X}^\tau_j \neq 1$, where $\hat{X}^\tau_i = X^\tau_i$; otherwise $\mathcal{SIM}$ sets sid^τ_i as in the protocol, computes

$$k^\tau_i = (c^\tau_{i,i-1})^n \cdot (\hat{X}^\tau_i)^{n-1} \cdot (\hat{X}^\tau_{i+1})^{n-2} \cdots (\hat{X}^\tau_{i-2}) \pmod{\hat{q}} \tag{1}$$

and outputs $K^\tau_i = H(k^\tau_i, \mathsf{sid}^\tau_i)$.

Reveal queries. On $\mathsf{Reveal}(U_i, \tau)$, if instance Π^τ_i has output a session key K^τ_i, $\mathcal{SIM}$ delivers it to $\mathcal{A}$.

Test query. Finally, if adversary issues command $\mathsf{Test}(i, \tau)$ then $\mathcal{SIM}$ picks a random bit b as $\mathcal{C}$ does, and if $b = 1$ then $\mathcal{SIM}$ replies with K^τ_i to $\mathcal{A}$. Otherwise, $\mathcal{SIM}$ returns a random value in $\{0,1\}^\kappa$.

Computing the RSA challenge. Every time $\mathcal{A}$ makes a query (r, s) to $H_{\hat{q}}$, $\mathcal{SIM}$ attempts to solve its RSA challenge as follows. For each entry $[j, \Pi^\tau_i, s^\tau_i, (\hat{\theta}^\tau_j, \hat{id}^\tau_j, \hat{\mu}^\tau_j), c^\tau_{i,j}]$ in table $T_{H_{\hat{q}}}$ s.t. $s^\tau_i = s$, $\mathcal{SIM}$ wants to check if

$$r = ((\hat{\theta})^{2e}(\hat{h})^{-2})^t = (\hat{\theta})^{2e(\gamma+d)}(a^e/g)^{-2(\gamma+d)} = (\hat{\theta}/a)^{2(e\gamma+1)} g^{2\gamma} g^{2d} \pmod{n} \tag{2}$$

where $\hat{\theta} = \hat{\theta}^\tau_j$, a is the value s.t. $\hat{h} = H_n(\hat{id}^\tau_j) = a^e \cdot g^{-1} \pmod{n}$, and $(t, \gamma) = (t^\tau_i, \gamma^\tau_i)$ defined when session Π^τ_i was started. Note that if $r = (\hat{z}^\tau_j)^{t^\tau_i}$, i.e. $\mathcal{A}$ queries $H_{\hat{q}}$ on pair $(r, s) = ((\hat{z}^\tau_j)^{t^\tau_i}, s^\tau_i)$, where $\hat{z}^\tau_j = (\hat{\theta}^\tau_j)^{2e}(\hat{h})^{-2}$ and t^τ_i is the value that satisfies $\bar{\theta}^\tau_i = (-1)^{b^\tau_i}(h_i)^d g^{t^\tau_i}$, then $r = ((\hat{\theta}^\tau_j)^{2e}(\hat{h})^{-2})^{t^\tau_i}$.

The way $\mathcal{SIM}$ can verify if equation (2) holds is to compute

$$w = r \cdot (\hat{\theta}/a)^{-2(1+e\gamma)} g^{-2\gamma} \pmod{n} \tag{3}$$

and test if $w^e = g^2$. If this holds then $\mathcal{SIM}$ extracts g^d by computing $w^\beta g^\alpha$, where α, β satisfy $e\alpha + 2\beta = 1$.

PART II: ANALYSIS OF THE SIMULATION

First note that if the adversary $\mathcal{A}$ runs in time t then the running time t' of the above simulator $\mathcal{SIM}$ is dominated by $t + m * t_{kg} + q_s * q_H * t_{exp}$, where t_{kg} is the time to generate an RSA private/public key pair and t_{exp} is the time of an exponentiation modulo n.

Denote as N_b the real network as executed by the challenger $\mathcal{C}$ with a fixed bit b. Recall that if $b = 0$ then $\mathcal{C}$ sends to $\mathcal{A}$ a random κ-bit long value and if $b = 1$ then $\mathcal{C}$ delivers the session key of the tested instance. We also denote as $\mathcal{SIM}_b$ an execution of the above simulator $\mathcal{SIM}$ with a fixed bit b on challenge (n, e, g) where g satisfies $\langle -1 \rangle \times \langle g \rangle = Z_n^*$.

We define the following events:

$\mathsf{NE_b}$: $\mathcal{A}$ outputs 1 on interaction with N_b.
$\mathsf{NE_{G,b}}$: $\mathcal{A}$ outputs 1 and tests session Π_i^τ s.t. $\mathsf{Group}(\Pi_i^\tau) = G$, on interaction with N_b.
$\mathsf{SE_{G,b}}$: $\mathcal{A}$ outputs 1 and tests session Π_i^τ s.t. $\mathsf{Group}(\Pi_i^\tau) = G$, on interaction with $\mathcal{SIM}_b$.
sCollision: There is a user U_i s.t. $s_i^{\tau_1} = s_i^{\tau_2}$ for some $\tau_1 \neq \tau_2$, either in an execution or in a simulation.
$\mathsf{H_n Failure}$: $\mathcal{A}$ queries H_n on id_i for some $U_i \in \mathsf{Rev}$ *before* this value is chosen, by $\mathcal{C}$ in an execution and by $\mathcal{SIM}$ in a simulation.
$\bar{\mathsf{NE}}_{G^*,b} = \mathsf{NE}_{G^*,b} \wedge \neg(\mathsf{H_n Failure} \vee \mathsf{sCollision})$
$\bar{\mathsf{SE}}_{G^*,b} = \mathsf{GE}_{G^*,b} \wedge \neg(\mathsf{H_n Failure} \vee \mathsf{sCollision})$
$\mathsf{H_{\hat{q}} Query}$: There is a session Π_i^τ s.t. $\mathcal{A}$ queries $H_{\hat{q}}$ on pair $(\hat{z}_j^\tau)^{t_i^\tau}, s_i^\tau)$, for $j = i-1$ or $j = i+1$, which relates to this Π_i^τ session, i.e. $\hat{z}_j^\tau = (\hat{\theta}_j^\tau)^{2e}(H_n(\hat{id}_j^\tau))^{-2}$, and t_i^τ satisfies $g^{t_i^\tau} = (\theta_i^\tau)^{2e}(H_n(id_i))^{-2}$.

Note that by the assumption that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sec}} = |Pr[b' = b]| \geq \epsilon$ we have $|\Pr[\mathsf{NE}_1] - \Pr[\mathsf{NE}_0]| \geq 2\epsilon$. Also, since $\Pr[\mathsf{E_b}] = \sum_{G \in \mathcal{G}} \Pr[\mathsf{NE_{G,b}}]$, let $G^* \in \mathcal{G}$ be a group s.t.

$$|\Pr[\mathsf{NE}_{G^*,1}] - \Pr[\mathsf{NE}_{G^*,0}]| \geq 2\epsilon/m \tag{4}$$

Assume that this is a group chosen by the simulator $\mathcal{SIM}$ above. (Note that $\mathcal{SIM}$ could also guess $\mathcal{G}^*$ with $1/m$ probability.) We will argue the following four facts:

$$|\Pr[\mathsf{NE}_{G^*,b}] - \Pr[\bar{\mathsf{NE}}_{G^*,b}]| \leq \Pr[\mathsf{H_n Failure} \wedge \mathsf{sCollision}] \text{ for } b = 0,1 \tag{5}$$

$$\Pr[\mathsf{H_n Failure} \wedge \mathsf{sCollision}] \leq l q_H 2^{-\kappa'} + q_s^2 \cdot 2^{-3\kappa} \tag{6}$$

$$|\Pr[\bar{\mathsf{SE}}_{G^*,1}] - \Pr[\bar{\mathsf{SE}}_{G^*,0}]| \leq q_H 2^{-\kappa'} \tag{7}$$

$$|\Pr[\bar{\mathsf{NE}}_{G^*,b} \mid \neg \mathsf{H_{\hat{q}} Query}] - \Pr[\bar{\mathsf{SE}}_{G^*,b} \mid \neg \mathsf{H_{\hat{q}} Query}]| \leq q_s 2^{-\kappa''+2} \text{ for } b = 0,1 \tag{8}$$

Note that by inequalities (4)-(5) it follows that for either $b = 0$ or $b = 1$ we have:

$$|\Pr[\bar{\mathsf{NE}}_{G^*,b}] - \Pr[\bar{\mathsf{SE}}_{G^*,b}]| \geq \epsilon/m - (l q_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_H 2^{-\kappa'}/2) \geq \epsilon/m - (2 l q_H 2^{-\kappa'} + q_s^2 2^{-3\kappa})$$

Together with (5), this inequality implies that

$$\Pr[\mathsf{H_{\hat{q}} Query}] \geq \epsilon/m - (2 l q_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_s 2^{-\kappa''+2})$$

Since whenever event $\mathsf{H_{\hat{q}} Query}$ happens the simulator $\mathcal{SIM}$ solves its RSA challenge, this implies our claim that $\epsilon' \geq \epsilon/m - (2 l q_H 2^{-\kappa'} + q_s^2 2^{-3\kappa} + q_s 2^{-\kappa''+2})$

It remains for us to argue that statements (5)-(5) above indeed hold. Note that inequality (5) follows immediately from the definition of $\mathsf{NE}_{\mathsf{G}^*,\mathsf{b}}$. For inequality (5) observe that $\Pr[\mathsf{H_n Failure}] \leq l q_H 2^{-\kappa'}$ because $|\mathsf{Rev}| \leq l$ and the response on each query to H_n is a random element in the set of size $\{0,1\}^{\kappa'}$. Also $\Pr[\mathsf{sCollision}] \leq q_s^2 \cdot 2^{-3\kappa}$ because a collision in s_i^τ values for any user U_i can only happen if two sessions $\Pi_i^{\tau_1}$ and $\Pi_i^{\tau_2}$ of this user choose the same value $\mu_i^{\tau_1} = \mu_i^{\tau_2}$. Since every session chooses its μ_i^τ value at random in a set of size $2^{-3\kappa}$, and there are at most q_s sessions, the above bound follows.

Equality (5) is also straightforward to see. First note that the statistical difference between all the values θ_i^τ in the execution and the simulation is $q_s 2^{-\kappa''+2}$, because for each Π_i^τ, the difference between distribution of t_i^τ chosen as in the execution as $t_i^\tau \leftarrow Z_{n/2}$, and the distribution of values $t_i^\tau = \gamma_i^\tau + d \pmod{\phi(n)/2}$ for γ_i^τ uniform in $Z_{n/2}$ (recall that this is how value t_i^τ is defined in the simulation), is at most $2^{-\kappa''+2}$. Everything else in the execution and the simulation is distributed in the same way, provided g is correct and event $\mathsf{H_n Failure}$ does not happen, except for the way values $c_{i,j}^\tau$ are computed. Now, if $\mathsf{H_{\hat{q}} Query}$ does not happen, i.e. if for all sessions Π_i^τ, adversary $\mathcal{A}$ does not query the hash function $H_{\hat{q}}$ on the proper pair $(\hat{z}_j^\tau)^{t_i^\tau}, s_i^\tau)$ that corresponds to the Π_i^τ session, then the way c's are computed in the execution (as outputs of $H_{\hat{q}}$) and the way they are picked in the simulation (at random in $Z_{\hat{q}}^*$ except if two sessions are partnered) are the same from $\mathcal{A}$'s point of view. The reason that's the case is that the only case in the protocol execution when two sessions $\Pi_i^\tau, \Pi_{i'}^{\tau'}$ compute two c values on the same input is if $s_i^\tau = s_{i'}^{\tau'}$. But if there is no collisions in s values (event $\mathsf{sCollision}$) then this implies in particular that that the adversary re-routed messages of these two sessions between each other, and in this case the simulator $\mathcal{SIM}$ also makes the two c values equal to one another.

It remains to argue that inequality (5) holds. Note that the only difference in these two interactions is that in $\bar{\mathsf{SE}}_{\mathsf{G}^*,1}$ $\mathcal{A}$ gets key $K_i^\tau = H(k_i^\tau, s_i^\tau)$ on tested Π_i^τ, while in $\bar{\mathsf{SE}}_{\mathsf{G}^*,0}$ $\mathcal{A}$ gets a random κ-bit value instead of K_i^τ. Note that in $\mathcal{A}$'s interaction with $\bar{\mathsf{SE}}$, if we disregard for a moment the information $\mathcal{A}$ gets from queries to $H(k_{i'}^{\tau'}, s_{i'}^{\tau'})$ for any $\Pi_{i'}^{\tau'}$ (this information is contained in the answers of Test and Reveal queries), then value k_i^τ is hidden from $\mathcal{A}$ in an information-theoretic way, i.e. it's uniformly distributed in $Z_{\hat{q}}^*$ independently from everything else $\mathcal{A}$ sees. The reason that's the case is because, by equation (1), for each Π_i^τ, value k_i^τ is distributed independently from $\mathcal{A}$'s view as long as $c_{i,i-1}^\tau$ is independent from $\mathcal{A}$'s view. Disregarding $\mathcal{A}$'s queries to H, the only way value $c_{i,i-1}^\tau$ enters into the information $\mathcal{A}$ gets in the simulation is via $X_i^\tau = c_{i,i+1}^\tau / c_{i,i-1}^\tau \pmod{\hat{q}}$, where the $c_{i,i+1}^\tau$ is chosen independently from $c_{i,i-1}^\tau$, *except* if Π_i^τ is partnered by $\mathcal{A}$'s Send_1 commands with some other session $\Pi_{i'}^{\tau'}$ (see the three conditions on sessions Π_i^τ and $\Pi_{i'}^{\tau'}$ in the procedure for $\mathcal{SIM}$ on Send_1 query). In that case we have $c_{i,j}^\tau = c_{i',j'}^{\tau'}$ for some $j = i \pm 1$ and $j' = i' \pm 1$, and thus we have to ask if $c_{i,i-1}^\tau$ is still perfectly uniform given $X_i^\tau, X_{i'}^{\tau'}$. Let us call a pair $(\Pi_i^\tau, \Pi_{i'}^{\tau'})$ *related* if this is the case and assume $j = i+1$ and $j' = i'-1$ (in general there are three other cases for $\mathcal{A}$ to pair up these sessions, but the argument given here can be extended to this general case). Let $\Pi_{i_1}^{\tau_{i_1}}, ..., \Pi_{i_n}^{\tau_{i_n}}$ be sessions s.t. for each j, sessions $\Pi_{i_j}^{\tau_{i_j}}$ and $\Pi_{i_{j+1}}^{\tau_{i_{j+1}}}$ are related in the above way. However, even in this case, each variable $c_{i_j,i_{j-1}}$, taken by itself, is still uniformly distributed in $Z_{\hat{q}}^*$ (although *not* independently

from one another) given $\mathcal{A}$'s view $X_{i_1}^{\tau_{i_1}}, ..., X_{i_n}^{\tau_{i_n}}$, because each X_{i_1} sets one constraint between two c's but there are $n+1$ independently chosen c's involved.

Finally, let us put back the additional information related to any of these c_{i_j,i_j-1} values that $\mathcal{A}$ gets from hash function outputs $H(k_{i_j}^{\tau_{i_j}}, s_{i_j}^{\tau_{i_j}})$. Note that $\mathcal{A}$ gets to see these outputs from both its Reveal and Test queries, and $\mathcal{A}$ can query H to search for the matching value $k_{i_j}^{\tau_{i_j}}$ for any $\Pi_{i_j}^{\tau_{i_j}}$ in a chain of related sessions $\Pi_{i_1}^{\tau_{i_1}}, ..., \Pi_{i_n}^{\tau_{i_n}}$. Learning any such $k_{i_j}^{\tau_{i_j}}$ value implies learning the corresponding $c_{i_j,i_j-1}^{\tau_{i_j}}$ value, and together with $X_{i_1}^{\tau_{i_1}}, ..., X_{i_n}^{\tau_{i_n}}$ this leads to recovery of *all* values $c_{i_1,i_1-1}^{\tau_{i_1}}, ..., c_{i_n,i_n-1}^{\tau_{i_n}}$. However, $\mathcal{A}$ can only make q_H hash queries to H, and since each of these values is (individually) uniform in $Z_{\hat{q}}^*$, the ability to query H can leak information on any of these values with probability at most $q_h 2^{-\kappa'}$, because $\hat{q}$ is a κ'-bit prime. This implies the $q_h 2^{-\kappa'}$ bound on the distance between the two simulations, for $b=0$ and $b=1$. □

Theorem 2. *The AH-AGKE protocol defined by the above tuple* (Setup, KGen, Add, Revoke, Handshake) *is Affiliation-Hiding.*

Proof. (sketch) Since the id values are chosen independently of the group, the only values which can reveal something about the group membership of honest players are the θ values sent in Round 1 and the X values sent in Round 2. The simulator $\mathcal{SIM}$ required for the affiliation-hiding notion (see section 2.2) is very simple: On the $\mathsf{Start}(\hat{id})$ command it picks μ_i^τ at random as in the protocol, and θ_i^τ as a random bitstring of length $(2\kappa'' + \kappa)$, and sends back $(\theta_i^\tau, \hat{id}, \mu_i^\tau)$. Then on the $\mathsf{Send}_1(\hat{id}, \mathcal{M})$ command to the same Π_i^τ "instance" of the protocol, $\mathcal{SIM}$ picks X_i^τ at random in $Z_{\hat{q}}^*$ and sends back (X_i^τ, μ_i^τ).

The argument that the adversary cannot tell X_i values creates at random in the above game from the X_i values returned by the honest parties in the protocol, follows the same lines as the argument given in the proof of *security* of this protocol. In summary, the only way $\mathcal{A}$ can tell these X_i values from random is by querying the $H_{\hat{q}}$ hash function on an "appropriate" input $((\hat{z}_j^\tau)^{t_i^\tau}, s)$, for $j = i \pm 1$, related to this session (these values are well-defined in both the simulation and the execution). However, by the same argument as given in the security proof, if that happens with non-negligible probability than such adversary can be used to break the RSA problem.

What's new in this proof is that we must show that the distribution of value θ sent by an honest user in this protocol is statistically close to a uniform distribution on $(2\kappa'' + \kappa)$-bit strings, denoted $Z_{2^{2\kappa''+\kappa}}$. Recall that for all groups $G_1, ..., G_m$ we have $2\kappa'' = |n_1| = ... = |n_m|$. We use $U \approx_S V$ to denote that distribution U is statistically close to V in the sense that the difference between these distributions is at most $O(2^{-min(\kappa,\kappa'')})$.

As we noted in the construction, values $(-1)^b g^t \pmod n$ are uniformly distributed in Z_n^* for $(b,t) \leftarrow \{0,1\} \times Z_{2p'q'}$. Take any $h \in Z_n^*$ and $\sigma = h^d \bmod n$. Define a random variable $\bar{\theta}_{b,t} = (-1)^b g^t \sigma \bmod n$. Since multiplication by σ is a permutation in Z_n^*, we have

$$\{\bar{\theta}_{b,t}\}_{(b,t)\leftarrow\{0,1\}\times Z_{2p'q'}} \equiv Z_n^*$$

Since $Z_{n/2} \approx_S Z_{2p'q'}$, the above implies that

$$\{\bar{\theta}_{b,t}\}_{(b,t)\leftarrow\{0,1\}\times Z_{n/2}} \approx_S Z_n^*$$

Because the proportion of elements in Z_n which are divisible by p' or q' is $O(2^{-\kappa''})$, we have $Z_n^* \approx_S Z_n$. Therefore

$$\{\bar{\theta}_{b,t}\}_{(b,t)\leftarrow\{0,1\}\times Z_{n/2}} \approx_S Z_n$$

Finally, we can mask the modulus n of a random value $\bar{\theta}$ in Z_n by choosing random $k \leftarrow [0, ..., \lfloor 2^{2\kappa''+\kappa}/n \rfloor]$, and adding kn to $\bar{\theta}$ over integers:

$$\{\bar{\theta}_{b,t} + kn\}_{(b,t,k)\leftarrow\{0,1\}\times Z_{n/2}\times Z_{\lfloor (2^{2\kappa''+\kappa})/n \rfloor}} \approx_S \{0,1\}^{2\kappa''+\kappa}$$

Therefore the difference between the distribution of values θ_i^τ in the protocol execution and a simulation where these values are chosen uniformly among $(2\kappa'' + \kappa)$-bit strings, is about $2^{-min(\kappa,\kappa'')}$. Since there are q_s sessions, the total difference in the two views contributed by all the θ values is at most $q_s 2^{-min(\kappa,\kappa'')}$. □

5 Affiliation-Hiding AGKA Scheme Based on the Diffie-Hellman Problem

We present the DH-based AH-AGKA scheme. Due to space constraints we present only the scheme, a sketch of correctness, and the statements of theorems about its security and affiliation-hiding. The proofs will be included in a full version of this paper [13]. We note that the proof of security of this AH-AGKE protocol follows a similar logic to the proof of security of the RSA-based AH-AGKA protocol in the previous section, but it includes rewinding, which results in the factor q_H of security degradation in the reduction.

- Setup: The setup algorithm outputs the standard discrete logarithm parameters (p, q, g), i.e., primes p, q of size polynomial in κ, s.t. g is a generator of a subgroup in Z_p^* of order q. We also define hash functions $H_q : \{0,1\}^* \rightarrow Z_q$, $\bar{H}_q : \{0,1\}^* \rightarrow Z_q^*$, and $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$.
- KGen: The secret key is chosen as a random number $x \in Z_q$ and the public key is $y = g^x \pmod p$.
- Add: For any user U in the group, $\mathcal{CA}$ computes the certificate cert as a Schnorr signature [15] on an empty message under the key y, namely $\mathsf{cert} = (w, t)$ where $w = g^r \pmod p$, and $t = r + xH_q(w) \pmod q$, for random $r \leftarrow Z_q$. Note that (w, t) satisfies equation $g^t = wy^{H_q(w)} \pmod p$.
- Revoke: To revoke user U, the $\mathcal{CRL}$ is appended with (hash of) w, where (w, t) was U's certificate.
- Handshake: This is an AGKA protocol executed by some set $\Delta = \{U_1, ..., U_n\}$ of players. Each player U_i starts a session $\Pi_i^{\tau_i}$ for a (locally) fresh τ_i, on *some* inputs $(\mathsf{cert}_i, y, \mathcal{CRL}_i)$ s.t. y is some public key, $\mathsf{cert}_i = (w_i, t_i)$ is U_i's certificate for this public key y, i.e. $\mathsf{cert}_i \in \mathsf{Certs}(y)$, and $\mathcal{CRL}$ is the (hopefully recent) CRL for group $\mathsf{Group}(y)$. The Handshake protocol is in Figure 2 below.

The inputs of instance Π_i^τ of player U_i are $\mathsf{cert}_i = (w_i, t_i)$, y, and $\mathcal{CRL}_i$. Note that $g^{t_i} = w_i y^{H_q(w_i)}$.

[Round 1]: Player U_i picks $\mu_i \leftarrow \{0,1\}^{3\kappa}$, and broadcasts (w_i, μ_i)

- Assume that player U_i received n-1 messages (w_1, μ_1), ..., (w_{i-1}, μ_{i-1}), (w_{i+1}, μ_{i+1}), ..., (w_n, μ_n) in Round 1. (This is a simplification as in Figure 1. See footnote 4.)
  If any two messages contain the same value μ_j or the same value w_j, player U_i rejects.
- U_i sets $s = (y, \{w_j, \mu_j\}_{j=1,...,n})$.
- If $w_j \in \mathcal{CRL}_i$ for any j then U_i picks a random value X_i in Z_q and sets $\mathsf{reject} = T$. Otherwise, U_i computes $X_i = \bar{H}_q((z_{i+1})^{t_i}, s)/\bar{H}_q((z_{i-1})^{t_i}, s) \pmod q$ where $z_{i-1} = w_{i-1} y^{H_q(w_{i-1})}$ and $z_{i+1} = w_{i+1} y^{H_q(w_{i+1})}$.
  (Note that if (w_{i-1}, t_{i-1}) and (w_{i+1}, t_{i+1}) are certificates under key y then $z_{i\pm1} = g^{t_{i\pm1}}$.)

[Round 2]: Player U_i broadcasts (X_i, μ_i).

- If in Round 2 player U_i receives n-1 values X_j accompanied by μ_j's that match the $\mu_1, ..., \mu_{i-1}, \mu_{i+1}, ..., \mu_n$ values above, and if $\mathsf{reject} \neq T$, then U_i computes $k_i = \bar{H}_q((z_{i-1})^{t_i}, s)^n \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i-2} \pmod q$ and outputs $K_i = H(k_i, \mathsf{sid}_i)$, where $\mathsf{sid}_i = (y, \{w_j, \mu_j, X_j\}_{j=1,...,n})$. Otherwise U rejects.

Fig. 2. DH-based Affiliation-Hiding AGKA protocol

Protocol Correctness. Similarly to the correctness argument for the RSA-based protocol, if n instances Π_i^τ are executed on the same public key y and their messages are properly exchanges, they output same values s_i^τ, sid_i^τ, and they all compute the same key material

$$k_i^\tau = \bar{H}_q(g^{t_{i-1}t_i}, s) * \bar{H}_q(g^{t_i t_{i+1}}, s) * ... * \bar{H}_q(g^{t_{i-2}t_{i-1}}, s) \bmod q$$

where t_i's are defined by the first message as in Figure 2. Therefore all partnered sessions also output the same keys K_i^τ.

Theorem 3. *The AH-AGKA scheme defined by the above tuple* (Setup, KGen, Add, Revoke, Handshake) *is affiliation-hiding.*

Theorem 4. *Assuming that the GSDH problem is (ϵ', t')-hard in the q-order subgroup generated by g in Z_p^*, the AH-AGKA scheme defined by* (Setup, KGen, Add, Revoke, Handshake) *above is $(\epsilon, t, q_s, q_H, l, m)$-secure in the Random Oracle Model for*

$$\epsilon = c_\epsilon * (\epsilon' + (ml q_H/q + q_s^2 2^{-3\kappa}))$$
$$t = c_t * (t'/q_H - (ml + q_H q_S) t_{\mathsf{exp}})$$

where t_{exp} is a cost of exponentiation in the subgroup generated by g and c_t, c_ϵ are small constants, assuming the cost of accessing the DDH oracle is constant.

References

[1] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H. Wong. Secret handshakes from pairing-based key agreements. In *24th IEEE Symposium on Security and Privacy*, Oakland, CA, May 2003.

[2] M. Bellare, A. Boldyreva, A. Desai and D. Pointcheval. Key-privacy in public-key encryption. In *Advances in Cryptology - ASIACRYPT 2001*, 2001.
[3] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key-exchange protocols. In *30th STOC'01*, 2001.
[4] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks In *Advances in Cryptology - EUROCRYPT 2000*, 2000.
[5] D. Boneh, H. Shacham, and B. Lynn. Short signatures from the Weil pairing. In *J. of Cryptology*, vol. 17, no. 4, pp. 297-319, 2004.
[6] E. Bresson, O. Chevassut, D. Pointcheval, and J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange In *Proceedings of the 8th ACM conference on Computer and communications security (CCS'01)*, 2001
[7] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology - EUROCRYPT 1994*, 1994.
[8] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology - CRYPTO 2001*, 2001.
[9] C. Castelluccia, S. Jarecki, and G. Tsudik. Secret handshakes from ca-oblivious encryption. In *Advances in Cryptology - ASIACRYPT 2004*, 2004.
[10] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private Information Retrieval In *Journal of the ACM*, Volume 45, Issue 6, Pages:965-981, 1998
[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router In *13th USENIX Security Symposium*, August 2004.
[12] S. Jarecki, J. Kim, and G. Tsudik. Authentication for Paranoids: Multi-Party Secret Handshakes. In *ACNS'06*, June 2006.
[13] S. Jarecki, J. Kim, and G. Tsudik. Group Secret Handshakes or Affiliation-Hiding Authenticated Group Key Agreement. To appear on IACR eprint archives (http://eprint.iacr.org), 2007.
[14] J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange In *Advances in Cryptology - ASIACRYPT 2003*. 2003
[15] C. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO 1989*, 1989.
[16] V. Shoup. On Formal Models for Secure Key Exchange. In *Theory of Cryptography Library*, 1999.
[17] G. Tsudik and S. Xu. A Flexible Framework for Secret Handshakes. In *Privacy-Enhancing Technologies Workshop (PET'06)*, June 2006. Earlier version appeared as a Brief Announcement in *ACM PODC'05*, August 2005.

Efficient Password-Authenticated Key Exchange Based on RSA*

Sangjoon Park, Junghyun Nam, Seungjoo Kim**, and Dongho Won

Information Security Group, School of Information and Communication Engineering, Sungkyunkwan University, Gyeonggi-do, 440-746, Korea
{sangjoon, jhnam, skim, dhwon}@security.re.kr

Abstract. In this paper, we propose an efficient password-authenticated key exchange (PAKE) based on RSA, called RSA-EPAKE. Unlike SNAPI using a prime pubic key e greater than an RSA modulus n, RSA-EPAKE uses the public key e of a 96-bit prime, where $e = 2H(n, s) + 1$ for some s. By the Prime Number Theorem, it is easy to find such an s. But the probability that an adversary finds n and s with $\gcd(e, \phi(n)) \neq 1$ is less than 2^{-80}. Hence, in the same as SNAPI, RSA-EPAKE is also secure against e-residue attacks. The computational load on Alice (or Server) and Bob (or Client) in RSA-EPAKE is less than in the previous RSA-based PAKEs such as SNAPI, PEKEP ,CEKEP, and QR-EKE. In addition, the computational load on Bob in RSA-EPAKE is less than in PAKEs based on Diffie-Hellman key exchange (DHKE) with a 160-bit exponent. If we exclude perfect forward secrecy from consideration, the computational load on Alice is a little more than that in PAKEs based on DHKE with a 160-bit exponent. In this paper, we compare RSA-EPAKE with SNAPI, PEKEP, and CEKEP in computation and the number of rounds, and provide a formal security analysis of RSA-EPAKE under the RSA assumption in the random oracle model.

1 Introduction

Key exchange protocols are cryptographic primitives which allow two communicating parties to share a common secure session key over an insecure channel. In general, two parties own predetermined secret keys with high entropy, which are stored in cryptographic devices like smart cards, USB tokens, etc. But, needs for cryptographic devices make cryptographic systems more complex and inconvenient. To overcome this drawback, human memorable passwords with low entropy are used instead of high entropy keys. But password based cryptographic systems have a weakness against off-line dictionary attacks (or password-guessing attacks) because passwords are generally drawn from a small space of possible passwords. In practice, it is not easy to design secure key exchange protocols

* This work was supported by the Korean Ministry of Information and Communication under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).

** Corresponding author.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 309–323, 2007.

based on passwords. This problem, which we call password-authenticated key exchange (PAKE), was first proposed by Bellovin and Merritt [3]. PAKE protocols are attractive for their simplicity, convenience, and security against off-line dictionary attacks. Since Bellovin and Merritt, a number of protocols for PAKE have been proposed. Most of PAKEs use public key primitives like RSA and Diffie-Hellman key exchange. In particular, Diffie-Hellman key exchange (DHKE) is suited for designing PAKE rather than RSA. Most of the well-known and secure PAKEs are based on Diffie-Hellman key exchange [1,4,5,7,8,9,10,11,18]. On the contrary, many of PAKEs based on RSA have been shown to be insecure [3,12,15]. Recently, there have been presented several PAKEs which are based on RSA and are secure under the RSA assumption in the random oracle model [13,19]. Unfortunately, they require two communicating parties to perform much more computation than PAKEs based on DHKE. In general, for perfect forward secrecy, RSA-based PAKEs have to generate an RSA modulus n in each session. By such a reason, the computational load on Alice (or Server) is much larger than in PAKEs based on DHKE. Even without considering perfect forward secrecy, Alice's load in RSA-based PAKEs are often larger than in PAKEs based on DHKE.

1.1 Related Works

In this subsection, we mainly summarize the previous results about PAKEs based on RSA. In 1992, Bellovin and Merritt first presented PAKEs using public key cryptosystems like RSA, ElGamal, and DHKE [3]. But RSA-based PAKE in Bellovin-Merritt's protocol is not secure against e-residue attacks as described in [3,15]. That is, by e-residue attacks, it is feasible to filter out incorrect passwords from a dictionary of possible passwords. They concluded that RSA is not suitable for designing PAKE. In 1997, Lucks proposed another RSA-based PAKE which was claimed to be secure against e-residue attacks [12].

In 2000, Mackenzie et al. showed that Lucks' protocol is also insecure against e-residue attacks [13]. Then, he proposed RSA-based password-authenticated key exchange protocol, called SNAPI, and provided a formal security proof under the RSA assumption in the random oracle model [13]. SNAPI is the first secure RSA-based PAKE. But, to ensure that a public key e is relatively prime to $\phi(n)$, Alice (Server) has to generate a prime e greater than n and Bob (Client) has to test if e is a prime greater than n. Since the generation of a large prime number and the primality test of a large prime number require massive computation, SNAPI is not practical.

In 2004, Zhang proposed two efficient RSA-based PAKEs, which are called PEKEP and CEKEP, and also provided a formal security proof under the RSA assumption in the random oracle model [19,20]. To avoid a public key e of a large prime in SNAPI, PEKEP and CEKEP uses RSA encryption keys e^{m_1} and e^{m_2}, respectively, where e is a small odd prime, $m_1 = \lceil \log_e n \rceil$, and $m_2 = \lceil \log_e \varepsilon^{-1} \rceil$ for $\varepsilon \leq 2^{-80}$. In PEKEP, Alice performs two RSA decryptions and Bob performs a single RSA encryption with a public key $e^{m_1} \approx n$. In CEKEP, Alice performs three RSA decryptions and Bob performs two RSA encryptions

with a public key $e^{m_2} \approx \varepsilon^{-1}$. Alice does not need to generate a large prime e and Bob does not have to test the primality of e. Thus, PEKEP and CEKEP are more efficient in computation than SNAPI. In case of $\varepsilon = 2^{-80}$, the computational load on Bob in CEKEP is lighter than in Diffie-Hellman based PAKEs. Although CEKEP reduces the computational load on Bob, it requires two additional flows than PEKEP and consists of 6 rounds. In general, most of PAKEs, including PEKEP and PAKEs based on DHKE, consist of 4 rounds. Zhang also proposed PAKE using quadratic residues, called QR-EKE, which is similar to PEKEP [21]. Instead of a small prime, $e = 2$ is used in QR-EKE and the encryption key is 2^t, where $t = \lceil \log_2 n \rceil$. Compared with PEKEP, QR-EKE reduces the computational load on Bob by 33% in average. But Bob's load is still more than in PAKEs based on DHKE.

Besides, there have been presented several RSA-based PAKEs that reduce the computational load on Alice or Bob [6,17,19,22]. But, they cause additional rounds or large communication overhead to verify the validity of the RSA public key.

1.2 Contributions

In this paper, we propose an efficient password-authenticated key exchange based on RSA, called RSA-EPAKE. Unlike SNAPI using a prime pubic key e greater than an RSA modulus n, RSA-EPAKE uses the public key e of a 96-bit prime, where $e = 2H(n, s) + 1$ for some s. By the Prime Number Theorem [16], it is easy to find such an s. But the probability that an adversary finds n and s with $\gcd(e, \phi(n)) \neq 1$ is less than 2^{-80}. Hence, in the same as SNAPI, RSA-EPAKE is also secure against e-residue attacks. We provide a formal security analysis of RSA-EPAKE under the RSA assumption in the random oracle model. In RSA-EPAKE, Alice (Server) has to generate a 96-bit prime together with an RSA modulus n and Bob (Client) needs to test primality for the 96-bit prime. In case of $n > 2^{1023}$, the computational load for generating a 96-bit prime is less than for a single RSA decryption and the computational load for the primality test of a 96-bit prime is less than for a single RSA encryption with a 80-bit exponent. From these, we can drive that the computational load in RSA-EPAKE is less than in SNAPI, PEKEP , CEKEP, and QR-EKE. RSA-EPAKE consists of only 4 rounds, which is less than 6 rounds of CEKEP. In addition, the computational load on Bob in RSA-EPAKE is even less than in PAKEs based on DHKE, but the computational load on Alice in RSA-EPAKE is larger than in PAKEs based on DHKE because Alice has to generate an RSA modulus n and a 96-bit prime e in each session. If we exclude perfect forward secrecy from consideration, we need not to generate them in each session. In this case, the computational load on Alice in RSA-EPAKE is a little more than that in PAKEs based on DHKE with a 160-bit exponent. On the contrary, the computational load on Alice in PEKEP, CEKEP, and QR-EKE is much larger than that in PAKEs based on DHKE.

The remainder of this paper is structured as follows. In Section 2, we introduce an overview of the security model and definitions for password-authenticated key exchange protocols. In Section 3, we describe RSA-EPAKE, an RSA-based protocol for efficient password-authenticated key exchange. In Section 4, we compare

RSA-EPAKE with the previous protocols in efficiency. Section 5 includes the formal security analysis of RSA-EPAKE. Finally, conclusions are discussed in Section 6.

2 Security Model and Definitions

We consider two-party protocol for key exchange. Let Alice and Bob be the two entities for key exchange protocol. They share a common weak password w in a password space $\mathcal{D}$ and generate a strong session key for protecting their communicated messages. To defeat their goal, there is an active adversary $\mathcal{A}$ who totally controls the communications between Alice and Bob. The adversary $\mathcal{A}$ may view, tamper with, deliver out of order, or refuse to deliver messages sent by the honest parties. $\mathcal{A}$ may also initiate concurrent executions of the protocol between Alice and Bob, and may attempt to impersonate one (or both) of the parties. Since the space $\mathcal{D}$ of possible passwords is small, $\mathcal{A}$ can enumerate all possible passwords and find out the correct password which matches the previous conversation. This type of attack is called an off-line dictionary attack or an off-line password guessing attack. Many of authentication and key exchange protocols, which are based on a password, have a weakness against off-line dictionary attacks. The security goal of password-authenticated key exchange is to design a cryptographic protocol which is secure against off-line dictionary attacks by an active adversary $\mathcal{A}$. To achieve the security goal, we introduce the formal security model for password-authenticated key exchange which is based on that of [1]. For completeness, we review main points of their definitions here, and refer the readers to [1] for more details.

Participants and Passwords. Let I denote the identities of the protocol participants. Each pair (A, B) of identities of I share a common secret password w which is randomly chosen from $\mathcal{D}$.

Adversarial Model. In the real world, a protocol determines how entities behave in response to input from their environment. In the formal model, these inputs are provided by the adversary. Each entity is assumed to be able to execute the protocol multiple times with different partners. Let Π_A^i denote the i-th instance of the entity A. All instance may be used only once. The adversary $\mathcal{A}$ can make queries to any instance and has unlimited access of Π_A^i oracles. In response of each query, an instance updates its internal state and returns its output to the adversary. Each instance has a session key sk, a session id sid, and a partner id pid. The query types are defined as follows:

- $\mathsf{Send}(A, i, M)$: The adversary $\mathcal{A}$ sends a message M to the instance Π_A^i. The instance executes as specified by the protocol and sends back its response to the adversary. Should the instance accept, this fact, as well as the sid and pid, will be made visible to the adversary. Should the oracle terminate, this too will be made visible to the adversary.

- $\mathsf{Execute}(A, i, B, j)$: This call carries out an honest execution between two instances Π_A^i and Π_B^j, where $A \neq B$ and instances Π_A^i and Π_B^j have not been used yet, and returns a transcript of that execution to the adversary.
- $\mathsf{Reveal}(A, i)$: The session key sk_A^i of Π_A^i is given to the adversary.
- $\mathsf{Test}(A, i)$: The instance Π_A^i generates a random bit b and outputs its session key sk_A^i to the adversary if $b = 1$, or else a random session key if $b = 0$. This query is allowed only once.
- $\mathsf{Oracle}(M)$: This gives the adversary oracle access to a function h, which is selected at random from some probability space Ω. In random-oracle model, h models a cryptographic hash function.

Partnering and Freshness. Let Π_A^i and Π_B^j be a pair of instances. We say that the instances Π_A^i and Π_B^j are partnered if: (1) both instances have accepted and have the same session id sid; and (2) the partner id of Π_A^i is B and the partner id of Π_B^j is A. In general, we let the session id sid be the ordered concatenation of all messages sent and received by the instance Π_A^i (or Π_B^j). We say that Π_A^i is fresh if: (1) it has accepted; and (2) an adversary $\mathcal{A}$ has not queried $\mathsf{Reveal}(A, i)$ or $\mathsf{Reveal}(B, j)$.

Correctness. If Π_A^i and Π_B^j are partnered and they are accepted, then they concluded with the same session key $sk_A^i = sk_B^j$.

Definitions of Security. Let Succ denote the event that $\mathcal{A}$ asks a single Test query on a fresh instance Π_A^i, and outputs $b' = b$, where b is the bit selected during the Test query. The advantage of an adversary $\mathcal{A}$ in an attacking protocol is defined as $\mathsf{Adv}(\mathcal{A}, P) = 2\Pr[\mathsf{Succ}] - 1$. Now, we will define the security of password-authenticated key exchange. A probabilistic polynomial time adversary $\mathcal{A}$ can check the validity of candidate passwords by on-line impersonation. We often call it an on-line dictionary attack. Since the password space $\mathcal{D}$ is finite, the adversary $\mathcal{A}$ can always find out the correct password by on-line dictionary attacks. Thus, a protocol is secure if an on-line dictionary attack is the best the adversary can do. Since the adversary can only test a single password in each on-line dictionary attack, the attack is bounded by the number of messages for on-line impersonation attacks, i.e., the number of Send queries.

Definition 1. *Let $|\mathcal{D}|$ denote the size of the password space $\mathcal{D}$ and let Q_{se} denote the number of Send queries to different instances. A password-authenticated key exchange (PAKE) protocol is secure if, for every polynomial time adversary $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{ake} \leq Q_{se}/|\mathcal{D}| + \epsilon$, where ϵ is negligible.*

3 RSA-EPAKE

In this section, we present an efficient password authenticated key exchange protocol based on RSA, which is simply called RSA-EPAKE. Before describing our protocol, we define some notations for our protocol. Let $\{0,1\}^k$ denote the

set of binary strings of length k and $\{0,1\}^*$ denote the set of binary strings of finite length. Let $\mathbb{Z}_n$ denote the set of non-negative integers less than n and let $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n : \gcd(x, \phi(n)) = 1\}$, where $\phi(\ )$ is the Euler's totient function. Define hash functions $H : \{0,1\}^* \to \{0,1\}^{k_1-1}$, $H_1, H_2, H_3 : \{0,1\}^* \to \{0,1\}^{k_2}$, and $h : \{0,1\}^* \to \mathbb{Z}_n$, where k_1, k_2, and ℓ are security parameters and n is an RSA modulus of ℓ bit. For example, we can consider $k_1 = 96$, $k_2 = 160$, and $\ell = 1,024$. We assume that H, H_1, H_2, H_3 and h are independent random functions. Let $\mathcal{D}$ be a password space and let Alice and Bob share a common password $w \in \mathcal{D}$. Now, we present our protocol RSA-EPAKE.

RSA-EPAKE

1. **Alice** randomly generates $r_1 \in \{0,1\}^{k_2}$ and RSA modulus n with $2^{\ell-1} < n < 2^\ell$. Next, she finds an s such that $e = 2H(n,s)+1$ is a k_1-bit prime and $\gcd(e, \phi(n)) = 1$. Finally, she sends (A, n, s, r_1) to Bob.
2. **Bob** randomly generates $r_2 \in \{0,1\}^{k_2}$, $a \in \mathbb{Z}_n^*$ and computes $\alpha = h(w, r_1, r_2, A, B, s, n)$ using the password w. Next, he tests if n is odd in $[2^{\ell-1}, 2^\ell]$, $e = 2H(n,s)+1$ is a k_1-bit prime, and $\gcd(\alpha, n) = 1$. If the test fails, then he rejects the protocol. Otherwise, he computes $c = a^e \cdot \alpha \bmod n$ and sends (B, r_2, c) to Alice.
3. **Alice** computes $\alpha = h(w, r_1, r_2, A, B, s, n)$. If $\gcd(\alpha, n) \neq 1$, she rejects the protocol. Otherwise, she computes $b = (c \cdot \alpha^{-1})^d \bmod n$, where $e \cdot d = 1 \bmod \phi(n)$, and $\mu = H_1(b, r_1, r_2, A, B, s, n)$. Finally, she sends (A, μ) to Bob.
4. **Bob** tests if μ is equal to $H_1(a, r_1, r_2, A, B, s, n)$. If not, then he rejects. Otherwise, he accepts and computes $\eta = H_2(a, r_1, r_2, A, B, s, n)$ and a session key $sk = H_3(a, r_1, r_2, A, B, s, n)$. Finally, he sends (B, η) to Alice.
5. **Alice**tests if η is equal to $H_2(b, r_1, r_2, A, B, s, n)$. If not, then she rejects. Otherwise, she accepts and computes a session key $sk = H_3(b, r_1, r_2, A, B, s, n)$.

In RSA-based PAKE, security against the e-th residue attacks has to be considered [3,12,15,13]. For an e-th residue attack, we assume that an adversary can intentionally choose (n, e) with $e|\phi(n)$, i.e., (n, e) is not a valid RSA public key. Next, the adversary selects a candidate password w_i drawn from $\mathcal{D}$ and computes $c' = c \cdot \alpha_i^{-1} \bmod n$, where c was received from Bob and $\alpha_i = h(w_i, r_1, r_2, A, B, s, n)$. If w_i is the correct password, then c' always has an e-th residue on modulo n. Otherwise, there is some possibility that c' has no e-th residue on modulo n. If c' has no e-th residue, then w_i is not the valid password. Since the adversary $\mathcal{A}$ generates n and knows $\phi(n)$, $\mathcal{A}$ can determine whether c' has an e-th residue or not. Thus, the adversary $\mathcal{A}$ can filter out invalid passwords from the password space $\mathcal{D}$. Throughout several protocols, he can reduce the number of candidate passwords and, finally, can be able to determine only one real password w. On the contrary, if $\gcd(e, \phi(n)) = 1$, then c' always has an e-th residue regardless of the correctness of the guessed password w_i. In this case, the adversary is unable to distinguish between the correct password w and an invalid password w_i. To avoid such an e-th residue attack, SNAPI uses a prime e larger than n [13]. Since Bob checks whether e is a prime larger than

Alice(A)		**Bob**(B)
password : w		password : w
Generate $n = p \cdot q \in [2^{\ell-1}, 2^{\ell}]$ and $r_1 \in_R \{0,1\}^{k_2}$		
Find s such that k_1-bit prime $e = 2H(n,s)+1$, and $\gcd(e, \phi(n)) = 1$	(A, n, s, r_1) $- - - - - \rightarrow$	$e = 2H(n,s)+1$ $r_2 \in_R \{0,1\}^{k_2}$, $a \in_R \mathbb{Z}_n^*$ $\alpha = h(w, r_1, r_2, A, B, s, n)$ If n isn't odd in $[2^{\ell-1}, 2^{\ell}]$, e isn't a k_1-bit prime, or $\gcd(\alpha, n) \neq 1$, then reject
	(B, r_2, c) $\leftarrow - - - - -$	$c = a^e \cdot \alpha \bmod n$
$\alpha = h(w, r_1, r_2, A, B, s, n)$ $d = e^{-1} \bmod \phi(n)$ If $\gcd(\alpha, n) \neq 1$, then reject $b = (c \cdot \alpha^{-1})^d \bmod n$ $\mu = H_1(b, r_1, r_2, A, B, s, n)$		
	(A, μ) $- - - - - \rightarrow$	$\mu \stackrel{?}{=} H_1(a, r_1, r_2, A, B, s, n)$ If not, reject $\eta = H_2(a, r_1, r_2, A, B, s, n)$
	(B, η) $\leftarrow - - - - -$	$sk = H_3(a, r_1, r_2, A, B, s, n)$
$\eta \stackrel{?}{=} H_2(b, r_1, r_2, A, B, s, n)$ If not, reject $sk = H_3(b, r_1, r_2, A, B, s, n)$		

Fig. 1. RSA-EPAKE Protocol

n, the adversary can not choose an RSA public key (n, e) with $e|\phi(n)$. But, this requirement imposes too much computation overhead on Alice and Bob. In our protocol RSA-EPAKE, we uses an RSA public key (n, e), where $e = 2H(n, s)+1$ is a k_1-bit prime for some s. To ensure the security against e-th residue attacks, we will prove that the probability of $\gcd(n, e) \neq 1$ is negligible. First, we introduce the Prime Number Theorem, which is also known as the Conjecture of Gauss and Legendre.

The Prime Number Theorem [16]: Let $\pi(x)$ denote the number of primes p with $0 < p < x$. Then, the asymptotic equation $\pi(x) \approx \lim_{x\to\infty} \frac{x}{\ln x}$ holds.

By the Prime Number Theorem, the probability that some $0 < p < x$ is a prime is $\frac{\pi(x)}{x} \approx \frac{1}{\ln x} \geq \frac{1}{\log_2 x}$. If $x = 2^{k_1}$, then $\frac{\pi(x)}{x} \geq \frac{1}{k_1}$. Thus, it is easy to find a k_1-bit prime e. But, the following Lemma says that it is hard for an adversary $\mathcal{A}$ to make an RSA public key (n, e) that e is a k_1-bit prime and a devisor of $\phi(n)$.

Lemma 1. *Let ℓ and k_1 be the security parameters and n be an RSA modulus of ℓ bit. Let $H : \{0,1\}^* \to \{0,1\}^{k_1-1}$ be a random oracle function. Then, for any probabilistic polynomial time algorithm $\mathcal{A}$, the following inequality holds:*

$$\mathsf{Adv}^H_{\mathcal{A}}(\ell, k_1) = \Pr\left[\begin{array}{l} e = 2H(n,s)+1 \text{ is a } k_1\text{-bit prime and} \\ \gcd(e, \phi(n)) \neq 1 \ : \ (n,s) \leftarrow \mathcal{A}(1^\ell, 1^{k_1}) \end{array}\right] < \frac{Q_h \cdot \ell}{2^{k_1-1}},$$

where Q_h is the number of random oracle calls.

Proof. Let the probability P be defined by

$$P = \Pr\left[\begin{array}{c} \gcd(e, \phi(n)) \neq 1 \quad : \quad \text{choose } (n,e) \text{ that} \\ n \text{ is a } \ell \text{ bit integer and } e \text{ is a } k_1\text{-bit prime} \end{array}\right].$$

By the Prime Number Theorem, the number of primes in $[2^{k_1-1}, 2^{k_1}]$ is greater than or equal to $\frac{2^{k_1-1}}{k_1}$, and the number of k_1-bit prime factors of $\phi(n)$ is less than or equal to $\lfloor \frac{\ell}{k_1} \rfloor$. Thus, $P \leq \lfloor \frac{\ell}{k_1} \rfloor \cdot \frac{k_1}{2^{k_1-1}} \leq \frac{\ell}{2^{k_1-1}}$. Since H is a random oracle function, a k_1-bit prime of type $e = 2H(n,s)+1$ is indistinguishable from a random prime of k_1 bit. Thus, for every adversary $\mathcal{A}$, $\mathsf{Adv}^H_{\mathcal{A}}(\ell, k_1) = Q_h \cdot P < \frac{Q_h \cdot \ell}{2^{k_1-1}}$. This completes the proof. □

4 Comparisons

In this section, we compare our protocol RSA-EPAKE with the previous protocols such as SNAPI, PEKEP and CEKEP [13,19]. Table 1 shows what Alice and Bob have to do to obtain a common session key. In Table 1, n denotes an RSA modulus of ℓ bit, ε is less than 2^{-80}, and q denotes a small prime such as 3, 5, 7, etc. For comparison of complexity, we assume that the complexity of an exponentiation to the power e modulo n is $O(|e| \cdot |n|^2)$, where $|x|$ denotes the bit size of x. In Table 1, "+" means an RSA encryption with a small prime public key such as 3, 5, 7, etc. This computation is too small and can be neglected.

First, we consider the computational burden on Alice (Server). In all protocols, Alice has to generate an RSA modulus n. Since generating an RSA modulus n implies generating two large primes of $\frac{\ell}{2}$ bits, this computational load is very large. In addition, Alice in SNAPI has to generate a prime e, which is greater than n, and Alice in RSA-EPAKE has to generate a 96-bit prime e. The computational load for generating a prime number greater than n is much larger than for generating an RSA modulus n. On the contrary, it is relatively very easy to generate a 96-bit prime. In case of $n > 2^{1023}$, the computational load for a 96-bit prime generation is even less than for a single RSA decryption. Thus, the computational load on Alice in RSA-EPAKE is less than in PEKEP and CEKEP even if Alice in PEKEP and CEKEP can easily obtain a public key e.

Now, we consider the computational burden on Bob (Client). In SNAPI and RSA-EPAKE, Bob has to test whether a given number e is a prime or not. The computational load for the primality test of an ℓ (≥ 1024) bit number is also very large, but the computational load for the primality test of a 96-bit number is even

Table 1. Comparisons between PAKEs based on RSA ((n, e) : RSA public key)

	SNAPI	PEKEP	CEKEP	RSA-EPAKE
Server Computations				
- Generation of n	○	○	○	○
- Generation of e	ℓ-bit prime	$q^{\lceil \log_q n \rceil}$	$q^{\lceil \log_q \varepsilon^{-1} \rceil}$	k_1-bit prime
- No. of decryptions	1	2	3	1
Client Computations				
- Primality test of e	ℓ-bit prime	-	-	k_1-bit prime
- No. of encryptions	1	1^+	2^+	1
- Complexity of encryption	$O(\ell^3)$	$O(\ell^3)$	$O(\ell^2 \log_2 \varepsilon^{-1})$	$O(\ell^2 k_1)$
No. of Rounds	4	4	6	4

(q is a small prime and, for real applications, $\ell = |n| = 1024$, $\varepsilon = 2^{-80}$, $k_1 = 96$)

less than for a single RSA encryption with an 80 bit public key. The primality test by the Miller-Rabin method requires only about 6 exponentiations (see Table 4.3 in [14]). For example, in RSA-EPAKE, the computational complexity for the primality test of a 96-bit prime e is about $6 \times O(96^3)$ and, in CEKEP with $\ell = 1024$ and $\epsilon \approx 2^{-80}$, the complexity of a single RSA encryption is about $O(\ell^2 \log_2 \varepsilon^{-1}) \approx O(1024^2 \times 80)$. In SNAPI with $\ell = 1024$, the complexity for a single RSA encryption is about $O(\ell^3) \approx O(1024^3)$, which is even greater than for two RSA encryptions in CEKEP. Thus, we can summarize the results of Table 1 as follows:

- Computational load on Alice :
  RSA-EPAKE < PEKEP < CEKEP < SNAPI
- Computational load on Bob :
  RSA-EPAKE < CEKEP < PEKEP < SNAPI

In the Diffie-Hellman based PAKE variant [1,5,7,8,9,10,18], both of Alice and Bob need to compute two modular exponentiations, each having an exponent of at least 160 bit. For perfect forward secrecy, Alice in RSA-EPAKE has to generate n and e in each session. RSA-EPAKE is more efficient in the computational load on Bob than the Diffie-Hellman based PAKE variant, but very inefficient in the computational load on Alice. For having trade-off between security and efficiency, we can consider that Alice generates them once an hour or once a day rather than in each session. In these cases, Alice performs only a single RSA decryption using CRT (Chinese Remainder Theorem) in each session. We performed experiments under Linux Fedora Core 4 on 2.4 GHz Pentium 4 processor. In the experiments, a single RSA decryption with 1024-bit n requires about $10,773 \times 10^3$ clock cycles and the exponentiation of $g^x \bmod p$, where $|g| = 32$, $|x| = 160$, and $|p| = 1024$, requires about $4,803 \times 10^3$ clock cycles. Hence, Alice's load in Diffie-Hellman based PAKE variant with a 160-bit exponent is about

90% of that in RSA-EPAKE. On the contrary, the computational loads on Alice in PEKEP and CEKEP is much larger than in Diffie-Hellman based PAKE variant. For our experiments, we used the Crypto Library of OpenSSL .

5 Formal Security Analysis

In this section, we analyze the security of RSA-EPAKE under the formal security model in Section 2. The security proof is almost similar to that of Zhang's PEKEP [19,20]. Thus, we will refer to some results and definitions from Zhang's full paper [20].

RSA Assumption: Let ℓ be the security parameter of RSA. Let key generator GE define a family of RSA functions, i.e., $(e, d, n) \leftarrow GE(1^\ell)$, where n is the product of two primes of the same size, $\gcd(e, \phi(n)) = 1$, and $e \cdot d = 1 \bmod \phi(n)$. For any probabilistic polynomial time algorithm $\mathcal{C}$, the following probability is negligible:

$$\mathsf{Adv}_{\mathcal{C}}^{rsa}(\ell) = \Pr\left[\begin{array}{c} x^e = c \bmod n : \\ (e, d, n) \leftarrow GE(1^\ell), c \in_R \{0,1\}^\ell, x \leftarrow \mathcal{C}(1^\ell, c, e, n) \end{array}\right]$$

Let $\mathsf{Adv}^{rsa}(\ell) = \max_{\mathcal{C}}\{\mathsf{Adv}_{\mathcal{C}}^{rsa}(\ell)\}$. Then, under the RSA assumption, the following theorem holds.

Theorem 1. *For any polynomial time adversary $\mathcal{A}$, the following inequality holds:*

$$Adv(\mathcal{A}, P_0) \leq \frac{Q_{se}}{|\mathcal{D}|} + (Q_{ex} + 3Q_{se})Adv^{rsa}(\ell) + \mathcal{O}(\frac{Q_{se} \cdot Q_h \cdot \ell}{2^{k_1 - 3}})$$

where $Q_{se} \leq |\mathcal{D}|$ denotes the number of Send *queries to different instances, Q_{ex} denotes the number of* Execute *queries, and Q_h denotes the number of random oracle calls.*

To prove the theorem, we describe five hybrid experiments by P_0, P_1, P_2, P_3, and P_4. Let $\mathsf{Adv}(\mathcal{A}, P_i)$ denote the advantage of $\mathcal{A}$ when participating in experiment P_i.

Hybrid experiment P_0: This describes the real adversary attack. During the attack, the adversary $\mathcal{A}$ makes a number of oracle calls (i.e., Send, Execute, Reveal, and Test). In addition, the adversary $\mathcal{A}$ has access to five independent random oracles H, H_1, H_2, H_3 and h. Each random oracles have a list of input-output pairs. If a new input of the random oracle was queried before, then the oracle returns the output in the list. Otherwise, the oracle generates a random number and returns it.

Hybrid Experiment P_1**:** In this experiment, the Execute oracle is modified so that the session keys of instances for which Execute is called are all chosen uniformly at random, that is, if the oracle $\mathsf{Execute}(A, i, B, j)$ is called between two instances Π_A^i and Π_B^j, then the session keys sk_A^i and sk_B^j are set equal to a random selected from $\{0,1\}^{k_2}$.

Before we present the experiments P_2, P_3, and P_4, we describe Send oracles which an active adversary $\mathcal{A}$ uses.

- $\mathsf{Send}_0(A, i)$: the instance Π_A^i generates an RSA modulus n in $[2^{\ell-1}, 2^\ell]$ and a random number $r_1 \in \{0,1\}^{k_2}$. Next, it finds an s such that $e = 2H(n, s) + 1$ is a k_1-bit prime and $\gcd(e, \phi(n)) = 1$. It returns A, n, s, and r_1 to the adversary.
- $\mathsf{Send}_1(B, j, A, n, s, r_1)$: the instance Π_B^j selects random numbers $r_2 \in \{0,1\}^{k_2}$ and $a \in \mathbb{Z}_n^*$. It queries the oracle h on $(w, r_1, r_2, A, B, s, n)$ and receives the reply α. If n isn't odd in $[2^{\ell-1}, 2^\ell]$, $e = 2H(n, s) + 1$ isn't a k_1-bit prime, or $\gcd(\alpha, n) \neq 1$, then Π_B^j rejects. Otherwise, Π_B^j computes $c = a^e \cdot \alpha \bmod n$ and returns r_2 and c to the adversary.
- $\mathsf{Send}_2(A, i, r_2, c)$: the instance Π_A^i obtains α as the reply of the oracle h on $(w, r_1, r_2, A, B, s, n)$. If $\gcd(\alpha, n) \neq 1$, then Π_A^i rejects. Otherwise, it computes $b = (c \cdot \alpha^{-1})^d \bmod n$, where $d = e^{-1} \bmod \phi(n)$, and obtains μ as the reply of H_1 on $(b, r_1, r_2, A, B, s, n)$. Then, it returns μ to the adversary.
- $\mathsf{Send}_3(B, j, \mu)$: the instance Π_B^j retrieves a, r_1, r_2, s, n from its internal states and queries the random oracle H_1 on $(a, r_1, r_2, A, B, s, n)$. If the reply of H_1 is not equal to μ, then it rejects. Otherwise, it computes η and sk_B^j as the replies of H_2 and H_3 on $(a, r_1, r_2, A, B, s, n)$, respectively. Then Π_B^j returns η to the adversary.
- $\mathsf{Send}_4(A, i, \eta)$: the instance Π_A^i retrieves b, r_1, r_2, s, n from its internal states and queries the random oracle H_2 on $(b, r_1, r_2, A, B, s, n)$. If the reply of H_2 is not equal to η, it rejects. Otherwise, it computes sk_A^i as the reply of H_3 on $(a, r_1, r_2, A, B, s, n)$.

A message is said to have been oracle-generated if it was an output by an instance; otherwise, it is said to have been adversarially-generated. If an instance Π_A^i receives a Π_B^j-oracle-generated message (r_2, c) in a Send_2 oracle call but the message (A, n, s, r_1) received by Π_B^j in a Send_1 oracle call was not generated by Π_A^i, we treat the message (B, r_2, c) as adversarially-generated.

Hybrid Experiment P_2**:** In this experiment, an instance Π_B^j receives a Π_A^i-oracle-generated message (A, n, s, r_1) in a Send_1 oracle call. If both instances Π_B^j and Π_A^i accept, they are given the same random session keys $sk \in_R \{0,1\}^{k_2}$, and if Π_B^j accepts but Π_A^i does not accept, then only Π_B^j receives a random session key and no session key is defined for Π_A^i.

Hybrid Experiment P_3**:** In this experiment, an instance Π_A^i receives a Π_B^j-oracle-generated message (B, r_2, c) in a Send_2 oracle call, while the instance Π_B^j has received a Π_A^i-oracle-generated message (A, n, s, r_1) in a Send_1 oracle call. If Π_A^i and Π_B^j accept, they are given the same session key $sk \in \{0,1\}^{k_2}$.

Hybrid Experiment P_4**:** In this experiment, we consider an instance Π_A^i (or Π_B^j) that receives an adversarially-generated message in a Send_2 (or Send_1) oracle call. In this case, if Π_A^i (or Π_B^j) accepts, then the experiment is halted and the adversary is said to have succeeded. This certainly improves the probability of success of the adversary.

Claim 1. For every polynomial-time adversary $\mathcal{A}$ making Q_{ex} oracle calls of type $\mathsf{Execute}$,

$$|\mathsf{Adv}(\mathcal{A}, P_1) - \mathsf{Adv}(\mathcal{A}, P_0)| \leq Q_{ex}\mathsf{Adv}^{rsa}(\ell) + Q_{ex}Q_h/\phi(n).$$

Claim 2. For every polynomial-time adversary $\mathcal{A}$ making Q_{se} oracle calls of type Send,

$$|\mathsf{Adv}(\mathcal{A}, P_2) - \mathsf{Adv}(\mathcal{A}, P_1)| \leq Q_{se}\mathsf{Adv}^{rsa}(\ell)$$
$$\mathsf{Adv}(\mathcal{A}, P_3) = \mathsf{Adv}(\mathcal{A}, P_2), \quad \mathsf{Adv}(\mathcal{A}, P_3) \leq \mathsf{Adv}(\mathcal{A}, P_4).$$

It is clear that $\mathsf{Adv}(\mathcal{A}) = \mathsf{Adv}(\mathcal{A}, P_0)$. Claim 1 and 2 are proved in the same way as Claim 1, Claim 2, Claim 3, and Claim 4 in Zhang's full paper [20]. The proofs are omitted due to lack of space. The following Claim 3 shows that the adversary's success probability in the experiment P_4 is negligible.

Claim 3. For every polynomial-time adversary $\mathcal{A}$ making $Q_{se} \leq |\mathcal{D}|$, queries to the Send_1 and Send_2 oracles,

$$\mathsf{Adv}(\mathcal{A}, P_4) \leq \frac{Q_{se}}{|\mathcal{D}|} + 2Q_{se}\mathsf{Adv}^{rsa}(\ell) + \frac{Q_{se} \cdot Q_h \cdot \ell}{2^{k_1-2}} + \frac{2Q_{se} \cdot Q_h}{\phi(n)} + \frac{Q_{se}}{2^{k_2-1}}.$$

Proof. Let Q_{se_1} and Q_{se_2} denote the number of Send_1 and Send_2 oracle calls made by the adversary in experiment P_4, respectively. We consider the following two cases:

[Case 1]: Consider an instance Π_A^i that receives an adversarially-generated message (B, r_2, c) in a Send_2 oracle. If $\gcd(\alpha, \phi(n)) = 1$, then Π_A^i rejects. So, we assume $\gcd(\alpha, \phi(n)) \neq 1$. Π_A^i computes $b = (c \cdot \alpha^{-1})^d \bmod n$, where $d = e^{-1} \bmod \phi(n)$, and returns $\mu = H_1(b, r_1, r_2, A, B, s, n)$ to the adversary $\mathcal{A}$. Now, the adversary $\mathcal{A}$ has to return η to the instance Π_A^i. Without knowledge of b, the probability for $\mathcal{A}$ to generate the correct η is just 2^{-k_2}. If $\mathcal{A}$ know b, then he can compute $\eta = H_2(b, r_1, r_2, A, B, s, n)$. Let p_b denote the probability that $\mathcal{A}$ can recover the integer b. Solving b from a random $c \in \mathbb{Z}_n^*$ implies the RSA decryption problem. Thus, we can prove that $p_b \leq \mathsf{Adv}^{rsa}(\ell) + Q_h/\phi(n)$(similar to the proof of Claim 1). In another way, $\mathcal{A}$ guesses a candidate password w_i and computes $c = a^e \cdot \alpha_i \bmod n$, where a is a random number in $\mathbb{Z}_n^*$ and $\alpha_i = h(w_i, r_1, r_2, A, B, s, n)$. If $\mathcal{A}$ guesses the correct password $w_i = w$, then $\alpha_i = \alpha$ and the congruence $x^e \cdot \alpha = a^e \cdot \alpha_i \bmod n$ has the unique solution a because of $\gcd(e, \phi(n)) = 1$. Hence, a is equal to b, computed from c by the

instance Π_A^i. If $\mathcal{A}$ guesses an invalid password $w_i \neq w$, then $\alpha_i \neq \alpha$ and c can be treated as a random number in $\mathbb{Z}_n^*$. Hence, we have

$$p_b \leq \Pr[\alpha_i = \alpha] + \mathsf{Adv}^{rsa}(\ell) + Q_h/\phi(n) \leq \frac{1}{|\mathcal{D}|} + \mathsf{Adv}^{rsa}(\ell) + Q_h/\phi(n)$$

$$\begin{aligned}\Pr[\mathsf{Succ}\ of\ Case\ 1] &\leq Q_{se_2}(p_b + 2^{-k_2}) \\ &\leq \tfrac{Q_{se_2}}{|\mathcal{D}|} + Q_{se_2}\mathsf{Adv}^{rsa}(\ell) + \tfrac{Q_{se_2}\cdot Q_h}{\phi(n)} + \tfrac{Q_{se_2}}{2^{k_2}}.\end{aligned}$$

[Case 2]: Consider an instance Π_B^j that receives an adversarially-generated message (A, n, s, r_1) in a Send_1 oracle. Π_B^j selects random numbers $r_2 \in \{0,1\}^{k_2}$, $a \in \mathbb{Z}_n^*$ and computes $\alpha = h(w, r_1, r_2, A, B, s, n)$. Then, Π_B^j tests if n is odd in $[2^{\ell-1}, 2^\ell]$, $e = 2H(n,s)+1$ is a k_1-bit prime, and $\gcd(\alpha, n) = 1$. If the test fails, then Π_B^j rejects the protocol. Otherwise, it returns r_2 and $c = a^e \cdot \alpha \bmod n$ to the adversary. Then, $\mathcal{A}$ has to return μ to Π_B^j. Without knowledge of a, the probability for $\mathcal{A}$ to compute the correct μ is just 2^{-k_2}. Let p_a denote the probability that $\mathcal{A}$ recovers the integer a and let AccPK denote the event that $\mathcal{A}$ generates public keys (n, s), where n is odd in $[2^{\ell-1}, 2^\ell]$ and $e = 2H(n,s)+1$ is a k_1-bit prime. These keys are accepted as correct public keys by the instance Π_B^j. Thus, we have

$$\begin{aligned}p_a &= \Pr[a|\mathsf{AccPK} \wedge \gcd(e, \phi(n)) = 1] \cdot \Pr[\mathsf{AccPK} \wedge \gcd(e, \phi(n)) = 1] \\ &\quad + \Pr[a|\mathsf{AccPK} \wedge \gcd(e, \phi(n)) \neq 1] \cdot \Pr[\mathsf{AccPK} \wedge \gcd(e, \phi(n)) \neq 1] \\ &\quad + \Pr[a|\neg\mathsf{AccPK}] \cdot \Pr[\neg\mathsf{AccPK}] \\ &\leq \Pr[a|\mathsf{AccPK} \wedge \gcd(e, \phi(n)) = 1] + \Pr[\mathsf{AccPK} \wedge \gcd(e, \phi(n)) \neq 1] \\ &\quad + \Pr[a|\neg\mathsf{AccPK}]\end{aligned}$$

If the event AccPK does not occur, then Π_B^j rejects the protocol. Thus, we can assume $\Pr[a|\neg\mathsf{AccPK}] = 0$. By Lemma 1, it is clear that $\Pr[\mathsf{AccPK} \wedge \gcd(e, \phi(n)) \neq 1] \leq \frac{Q_h \cdot \ell}{2^{k_1-1}}$. Now, let us consider the event $\mathsf{AccPK} \wedge \gcd(e, \phi(n)) = 1$. Then, $\mathcal{A}$ can compute $\alpha_i = h(w_i, r_1, r_2, A, B, s, n)$ using a guessing password w_i. The congruence $c = x^e \cdot \alpha_i \bmod n$ has a unique solution a' because of $\gcd(e, \phi(n)) = 1$. If $\mathcal{A}$ guesses the correct password $w_i = w$, then $\alpha_i = \alpha$ and the solution a' is equal to a, selected by the instance Π_B^j. Thus, we have

$$\Pr[a|\mathsf{AccPK} \wedge \gcd(e, \phi(n)) = 1] = \Pr[\lambda = \alpha] = \frac{1}{|\mathcal{D}|},$$

$$\Pr[\mathsf{Succ}\ of\ Case\ 2] \leq Q_{se_1}(p_a + 2^{-k_2}) = \frac{Q_{se_1}}{|\mathcal{D}|} + \frac{Q_{se_1} \cdot Q_h \cdot \ell}{2^{k_1-1}} + \frac{Q_{se_1}}{2^{k_2}}.$$

From the analysis in Case 1 and Case 2, the adversary' success probability in experiment P_4 is upper bounded by

$$\begin{aligned}\Pr[\mathsf{Succ}] &= \Pr[\mathsf{Succ}\ of\ Case\ 1] + \Pr[\mathsf{Succ}\ of\ Case\ 2] \\ &\leq \tfrac{Q_{se}}{|\mathcal{D}|} + Q_{se}\mathsf{Adv}^{rsa}(\ell) + \tfrac{Q_{se}\cdot Q_h\cdot\ell}{2^{k_1-1}} + \tfrac{Q_{se}\cdot Q_h}{\phi(n)} + \tfrac{Q_{se}}{2^{k_2}}\end{aligned}$$

where $Q_{se} = Q_{se_1} + Q_{se_2}$. Since $Q_{se}/|\mathcal{D}| \leq 1$, we have

$$\begin{aligned}\mathsf{Adv}(\mathcal{A}, P_4) &= 2\Pr[\mathsf{Succ}] - 1 \\ &\leq \frac{Q_{se}}{|\mathcal{D}|} + 2Q_{se}\mathsf{Adv}^{rsa}(\ell) + \frac{Q_{se}\cdot Q_h\cdot \ell}{2^{k_1-2}} + \frac{2Q_{se}\cdot Q_h}{\phi(n)} + \frac{Q_{se}}{2^{k_2-1}}\end{aligned}$$

This completes the proof of Claim 3. □

By combining Claims 1 to 3 and assuming $\phi(n) > 2^{k_2} > 2^{k_1}$, Theorem 1 holds:

$$\mathsf{Adv}(\mathcal{A}, P_0) \leq \frac{Q_{se}}{|\mathcal{D}|} + (Q_{ex} + 3Q_{se})\mathsf{Adv}^{rsa}(\ell) + \mathcal{O}(\frac{Q_{se}\cdot Q_h\cdot \ell}{2^{k_1-3}}).$$

If (A, n, s, r_1) and (r_2, c) are oracle-generated messages and μ (or η) is an adversarially-generated message in a Send_3 (or Send_4) oracle, then the adversary $\mathcal{A}$ can not know a (or b). Without knowledge of a (or b), the probability that $\mathcal{A}$ guesses the correct μ (or η) is 2^{-k_2} and is negligible. So, in the Hybrid experiment P_4, we excluded the previous two cases from consideration.

By the RSA assumption, $\mathsf{Adv}^{rsa}(\ell)$ is negligible and, for sufficiently large $k_1 \geq 96$, the probability $\frac{Q_{se}\cdot Q_h\cdot \ell}{2^{k_1-3}}$ is also negligible. Then, by Definition 1, the following theorem holds.

Theorem 2. *RSA-EPAKE is a secure password-authenticated key exchange protocol under the RSA assumption in the random oracle model.*

6 Conclusions

In this paper, we have proposed an efficient password-authenticated key exchange based on RSA which is called RSA-EPAKE. We have shown that the success probability of an e-residue attack against RSA-EPAKE is negligible and have provided a formal security proof under the RSA assumption in the random oracle model. RSA-EPAKE is more efficient in computation than any other PAKEs based on RSA and the computational load on Bob (Client) is less than in PAKEs based on Diffie-Hellman key exchange. Moreover, without consideration of forward secrecy, the computational load on Alice (Server) is a little more than that in PAKEs based on Diffie-Hellman key exchange.

References

1. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attack. *Proc. of Eurocrypt 2000, LNCS vol.1807, Springer-Verlag*, pp.139-155, 2000.
2. M. Bellare and P. Rogaway. Entity authentication and key distribution. *Proc. of Crypto'94, LNCS vol.950, Springer-Verlag*, pp.92-111, 1995.
3. S. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. *Proc. of the IEEE Symposium on Research in Security and Privacy*, pp.72-84, May 1992.
4. S. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. *Proc. of the 1st ACM Conference on Computer and Communications Security, ACM*, pp.244-250, Nov. 1993.

5. V. Boyko, P. MacKenzie, and S. Patel. Provably secure password authenticated key excahnge using Diffie-Hellman. *Proc. of Eurocrypt 2000, LNCS vol.1807, Springer-Verlag*, pp.156-171, 2000.
6. D. Catalano, D. Pointcheval, and T. Pornin. IPAKE: Isomorphism for password-based authenticated key exchange. *Proc. of Crypto 2004, LNCS vol.3152, Springer-Verlag*, pp.477-493, 2004.
7. R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. *Proc. of Crypto 2003, LNCS vol.2656, Springer-Verlag*, pp.524-542, 2003.
8. O. Goldreich and Y. Lindell. Session-key generation using human passwords only. *Proc. of Crypto 2001, LNCS vol.2139, Springer-Verlag*, pp.408-432, 2001.
9. D. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review, ACM SIGCOMM, vol. 26, no. 5*, pp.5-26, 1996.
10. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. *Proc. of Eurocrypt 2001, LNCS vol.2045, Springer-Verlag*, pp.475-494, 2001.
11. T. Kwon. Authentication and key agreement via memorable passwords. *Proc. of Network and Distributed System Security Symposium*, Feb., 2001.
12. S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. *Proc. of Security Protocol Workshop, LNCS vol.1361, Springer-Verlag*, pp.79-90, 1997.
13. P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. *Proc. of Asiacrypt 2000, LNCS vol.1976, Springer-Verlag*, pp.599-613, 2000.
14. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography. CRC Pres*, Oct. 1996.
15. S. Patel. Number theoretic attacks on secure password schemes. *Proc. of IEEE Symposium on Security and Privacy*, May 1997.
16. V. Shoup. *A Computational Introduction to Number Theory and Algebra, Cambridge University Press*, 2005.
17. D. Wong, A. Chan, and F. Zhu. More efficient password authenticated key exchange based on RSA. *Proc. of Indocrypt 2003, LNCS vol.2904, Springer-Verlag*, pp.375-387, 2003.
18. T. Wu. The secure remote password protocol. *Proc. of Network and Distributed System Security Symposium, Sandiego*, pp.97-111, Mar. 1998.
19. M. Zhang. New approaches to password authenticated key exchange based on RSA. *Proc. of Asiacrypt 2004, LNCS vol.3329, Springer-Verlag*, pp.230-244, 2004.
20. M. Zhang. New approaches to password authenticated key exchange based on RSA. *url=http://eprint.iacr.org, Cryptology ePrint Archive, Report 2004/033.*
21. M. Zhang. Password authenticated key exchange using quadratic residues. *Proc. of ACNS 2004, LNCS vol.3089, Springer-Verlag*, pp.233-247, 2004.
22. F. Zhu, D. Wong, A. Chan, and R. Ye. RSA-based password authenticated key exchange for imbalance wireless networks. *Proc. of Information Security Conference (ISC'02), LNCS vol.2433, Springer-Verlag*, pp.150-161, 2002.

Non-degrading Erasure-Tolerant Information Authentication with an Application to Multicast Stream Authentication over Lossy Channels*

Yvo Desmedt[1,**] and Goce Jakimoski[2]

[1] Department of Computer Science, University College London
Gower Street, London WC1E 6BT, United Kingdom
y.desmedt@cs.ucl.ac.uk
[2] Department of Electrical and Computer Engineering, Burchard 212,
Stevens Institute of Technology, Hoboken, NJ 07030, USA
Goce.Jakimoski@stevens.edu

Abstract. The concept of erasure-tolerant information authentication was recently introduced to study an unconditionally secure setting where it is allowed to lose a limited number of message letters during transmission. Even if a part of the message is lost, the verifier will still be able to check the authenticity of some or all of the received message letters. In general, there might be some letters whose authenticity cannot be verified although they have arrived at the recipient's side. These letters will be discarded.

We consider a special case when the verifier can always check the authenticity of all received message letters. This property is desirable since no data will be lost due to the verifier's inability to verify its authenticity (i.e., the scheme does not introduce additional degradation of the quality of the received information). We provide necessary and sufficient conditions for a set system based erasure-tolerant authentication scheme to be non-degrading. We also discuss efficient implementations and propose a provably secure stream authentication scheme that makes use of erasure-tolerant authentication codes.

Keywords: cryptography, message authentication, stream authentication, erasure-tolerant information authentication, combinatorics.

1 Introduction

1.1 Erasure-Tolerant Information Authentication

In the standard information authentication model, the messages are treated as atomic objects: The verifier cannot check authenticity of incomplete messages.

* This research was done while the authors were at Florida State University, sponsored by NSF CCR-0209092.

** The author is BT Professor of Information Security and a courtesy professor at the Department of Computer Science, Florida State University, USA.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 324–338, 2007.

The notion of erasure-tolerant message authentication was recently introduced [17] to analyze a somewhat different setting where it is too expensive to provide an erasure-free channel and the application allows a limited data loss (e.g., audio and video streams). In the erasure-tolerant authentication model, it is allowed parts of the transmitted message to be erased (lost). Even if some of the data is lost, the receiver should be able to verify the authenticity of a part or possibly the complete message.

In a general erasure-tolerant authentication code, it is possible that the verifier will not be able to check the validity of all received message letters (packets). The letters whose authenticity cannot be verified will be discarded, and thus, we will have erasures that are due not to the data loss on the communication channel, but to the inability of the receiver to verify the authenticity of some received letters. Since this quality degradation introduced by the erasure-tolerant authentication scheme is not desirable, we consider a special class of *non-degrading* erasure-tolerant authentication schemes. In these schemes, the authenticity of each received letter (packet) can be always checked.

1.2 Our Contribution

We provide necessary and sufficient conditions that link the non-degrading erasure-tolerant authentication schemes and the cover-free families in design theory. We also discuss efficient implementations and present an efficient and provably secure stream authentication scheme that is based on erasure-tolerant authentication codes.

1.3 Related Work

The notion of erasure-tolerant message authentication was formally introduced by Jakimoski [17]. A similar concept is that of content extraction signatures introduced by Steinfeld et al [31]. A related notion is also the notion of approximate message authentication codes, which was introduced by Graveman et al [16]. Di Crescenzo et al [9] provide a security model for a rigorous study of the approximate message authentication codes.

The theory of unconditional authentication was developed by Simmons [29,30]. He also derived some lower bounds on the deception probability. Stinson [32] studied the properties of authentication codes that have minimum possible deception probabilities and minimum number of encoding rules. Some other fundamental results concerning authentication codes can be found in [21,6,40,33,34,35,4,10,27,1].

Cover-free families were introduced by Kautz and Singelton [18] to investigate superimposed binary codes. Additional constructions, bounds and applications can be found in [12,13,25,36,37,11].

Gennaro and Rohatgi [14] have proposed a stream signing scheme based on a chain of one-time signatures. A similar scheme has been proposed by Zhang [42] for authentication in routing protocols. The major disadvantage of the scheme is its non-resilience to packet loss. Various improvements were proposed subsequently [8,2,41,28,5,38] culminating with the solutions presented in [24] (see

also [3]). Some different approaches and possible improvements of the schemes presented in [24] have been discussed in [22,23,7,39].

2 Preliminaries

2.1 The Erasure-Tolerant Message Authentication Model

There are three participants in the erasure-tolerant message authentication model proposed in [17]: a transmitter, a receiver and an adversary. The transmitter wants to send a source state (or plaintext) to the receiver using a public communication channel. It is assumed that all plaintexts are strings of length k whose letters are from some alphabet Q. The transmitter has a key source from which he obtains a key. Prior to the message being sent, the secret key is communicated to the receiver through a secure channel. The transmitter uses the secret key to encode the plaintext into a q-ary string of length n. The derived message (or ciphertext) is sent through the public channel. The receiver will use the secret key to verify the validity of the received message. If at most $t < n$ letters are missing from the original intact valid message and the positions of the missing letters within the message are known, then the received message is still considered valid. In this case, the receiver accepts a plaintext that is derived from the original plaintext by erasing at most r ($r < k$) letters. If the received message is not derived from some intact valid message by erasing at most t letters, then the receiver does not accept a plaintext.

It is assumed that the secret key will be used only once. Hence, there are only two types of threats: impersonation and substitution. In an impersonation attack, the adversary, based only on his knowledge of the authentication scheme, can send a fraudulent message to the receiver when in fact no message has yet been sent by the transmitter. In a substitution attack, the adversary can intercept one valid message and replace it with his fraudulent message. The probability of successful impersonation is defined as the probability of success when the adversary employs an optimum impersonation strategy. The probability of successful substitution is defined as the probability of success when the adversary employs an optimal substitution strategy. Finally, the adversary may be able to select whether to employ an impersonation or a substitution attack (a deception attack). The probability of successful deception is the probability of success when an optimum deception strategy is employed.

2.2 Cover-Free Families

A cover-free family is defined as follows.

Definition 1. [37] *Let X be a v-set and let $\mathcal{B}$ be a set of subsets (blocks) of X. The set system $(X, \mathcal{B})$ is called a (w,t)-cover-free family if for any w blocks $B_1, \ldots, B_w \in \mathcal{B}$ and any other t blocks $A_1, \ldots, A_t \in \mathcal{B}$, we have*

$$\bigcap_{i=1}^{w} B_i \not\subseteq \bigcup_{j=1}^{t} A_j. \tag{1}$$

Let $N(w,t,b)$ denote the minimum number of points in any (w,t)-cover-free family that has b blocks. The following bound can be found in [37]:

$$N(w,t,b) \leq \frac{(w+t)\log b}{-\log p}, \qquad \text{where} \qquad p = 1 - \frac{t^t w^w}{(t+w)^{t+w}}. \tag{2}$$

2.3 Multicast Stream Authentication

Most of the multicast stream authentication schemes are based on the concept depicted in Figure 1 (see [24]). To authenticate the packet (chunk) P_i of the stream, the sender first commits to the key value K_i by sending $H(K_i)$ in the packet P_{i-1}. The key K_i is only known to the sender, and it is used to compute a MAC on the packet P_i. After all recipients have received the packet P_i, the sender discloses the key value K_i in the packet P_{i+1}. The recipients verify whether the received key value corresponds to the commitment and whether the MAC of the packet P_i computed using the received key value corresponds to the received MAC value. If both verifications are successful, the packet P_i is accepted as authentic. Note that P_i contains the commitment to the next key value K_{i+1}. To bootstrap the scheme, the first packet, which contains the commitment $H(K_1)$ to the first symmetric key K_1, is signed using a digital signature scheme (e.g., RSA).

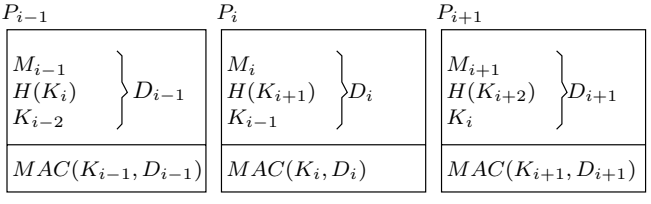

Fig. 1. Multicast stream authentication: The basic scheme

3 Erasure-Tolerant Signature Schemes

The model described in Section 2.1 can be extended to the case of digital signatures and MAC schemes by adapting the notions of existential forgery and unforgeability [15] to the setting where erasures are allowed.

The differences between an erasure-tolerant digital signature setting and the unconditionally secure setting described in Section 2.1 are:

- The sender uses the secret key to sign a large number of messages.
- The receiver uses a public key to verify the validity of the received messages.
- The adversary has access to the public key and can query a signing oracle many times. The goal of the adversary is to produce a forgery (i.e., a message that is accepted as valid by the receiver, but not derived from a previously signed message by erasing at most t letters).

– The computational power of the adversary is not unlimited. An erasure-tolerant digital signature scheme is considered unforgeable (secure) if there is no efficient adversary that can produce a forgery with non-negligible probability.

One can similarly define the erasure-tolerant MAC setting.

4 Non-degrading Erasure-Tolerant Message Authentication: Constructions Based on Cover-Free Families

Let us consider the following generic construction of erasure-tolerant unconditionally secure authentication codes or erasure-tolerant signature schemes. Given a large message M, the sender divides it into a sequence of b smaller messages (called letters earlier on) $M_1, \ldots, M_b$ and computes v authentication tags (resp., signatures) $\tau_1, \ldots, \tau_v$. Each tag τ_j is computed over a subsequence of the sequence of messages using a message authentication (or signature) scheme. We say that the authentication tag τ_j depends on the message M_i if M_i is in the subsequence which is used to compute τ_j. The sender then constructs and sends b packets $P_1, \ldots, P_b$. Each packet P_i includes the message M_i and all authentication tags τ_j with the following property: the message M_i is the last message in the subsequence that is used to compute τ_j.

We allow at most t of the b packets to be lost during the transmission. The recipient uses the received tags to verify the authenticity of the received messages. In particular, if all messages that are used to compute some authentication tag τ_j have arrived, the recipient uses the received τ_j to verify the authenticity of the subsequence. If the verifier outputs one, then the recipient accepts all the messages in the subsequence as valid.

Obviously, if a tag τ_j depends on a lost message M_i, then τ_j cannot be used to verify the authenticity of the rest of the messages in the subsequence that was used to compute τ_j. Hence, a situation might occur where the recipient will not be able to verify the authenticity of a message that was not lost. If the subsequences are chosen in such manner so that the receiver will be able to verify the authenticity of each message that is not lost, then we say the erasure-tolerant authentication code (resp., signature scheme) is *non-degrading*.

To analyze the problem of constructing non-degrading schemes, it is convenient to define a *dependency set system* $(X, \mathcal{B})$, where:

– $X = \{\tau_1, \ldots, \tau_v\}$
– $\mathcal{B} = \{B_j | B_j = \{\tau_i | \tau_i \text{ depends on } M_j\}, 1 \leq j \leq b\}$

If a message M_i is lost during the transmission, then the authentication tags in B_i become useless. Assume that $M_{i_1}, \ldots, M_{i_t}$ are the lost messages. Then, the set of all useless tags is $\bigcup_{l=1}^{t} B_{i_l}$ (see Figure 2). The authenticity of the message M_j can be verified if and only if $B_j \nsubseteq \bigcup_{l=1}^{t} B_{i_l}$. Hence, we have the following theorem.

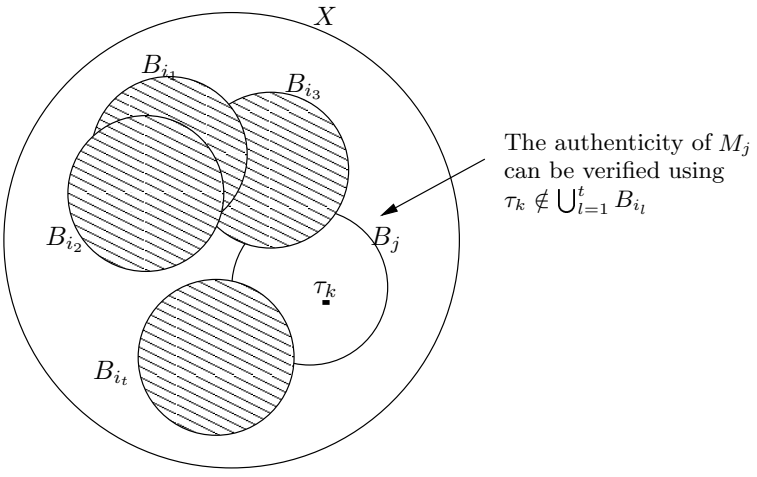

Fig. 2. Erasure-tolerant authentication using cover-free families

Theorem 1. *An erasure-tolerant authentication code (resp., signature scheme) constructed as above is non-degrading if and only if the corresponding dependency set system is a* $(1, t)$*-cover-free family.*

A simple example is given in Figure 3. One can easily verify that if one message is lost ($t = 1$), then we can still verify the authenticity of the rest of the packets. For example, assume M_1 is lost. Then, the authentication tags τ_1 and τ_5 are not usable anymore. However, the authenticity of M_2, M_3 and M_4 still can be verified using τ_6, τ_7 and τ_8 respectively, and the authenticity of M_5, M_9 and M_{13} can be verified using τ_2, τ_3 and τ_4 respectively.

Note that the number of tags in the example is eight instead of sixteen, which is the case when we authenticate each packet separately. In general, we have two extremes. The number of tags can be as low as a logarithm of the number of packets (see Inequality 2). The other extreme is to use one tag per message (i.e., the number of tags is equal to the number of messages). The main advantages of using a small number of tags / signatures are:

- The communication (or storage) overhead is smaller.
- When unconditionally secure message authentication is employed, we need a fresh key material for each tag. Hence, the key generation time decreases when the number of tags is smaller.
- In the case of digital signature schemes, the time complexity will be reduced since the number of signatures is smaller.

Remarks. Instead of the approach presented here, one can use a combination of authentication codes and erasure-resilient codes [19,20] resulting in full-recovery erasure-tolerant authentication codes [17]. The advantage of the second approach is that the sender will be able to recover the whole message M. The disadvantage is that at least t out of the b packets must be redundant. Moreover, it is commonly

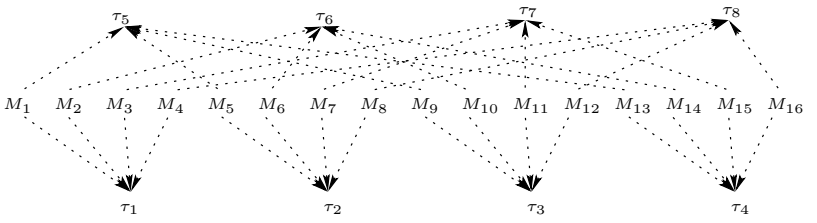

Fig. 3. $(1, 1)$-cover-free family

accepted that in the case of audio or video applications there will be some data loss. The loss-tolerance of TESLA [24] was one of the main reasons it was accepted as an Internet standard.

To simplify the description, we assumed that the messages arrive in order. However, it might be possible that the messages can be reordered so that the derived sequence is also valid. A malicious reordering can be prevented if the messages also include unique numbers identifying their position.

Also, note that more than t erasures can be tolerated in a scheme derived from a $(1, t)$-cover-free family. In particular, if more than t messages are lost, we can still verify the authenticity of some of the received messages. However, we cannot verify the authenticity of all received messages (i.e., non-degrading property does not hold). For example, in the scheme from Figure 3, even if three messages are lost, we will still be able to verify the validity of at least four messages.

5 Multicast Stream Authentication over Lossy Channels Using Erasure-Tolerant Primitives

In this section, we present a provably secure multicast stream authentication scheme that uses an authentication code instead of a MAC scheme to authenticate the packets of the stream. A loss-tolerant multicast stream authentication scheme can be obtained if one uses erasure-tolerant instead of regular authentication codes.

We first (see Section 5.1) discuss a basic method in which we are *not* dealing with erasures.

5.1 Multicast Stream Authentication Using Authentication Codes

Each key K_i in the basic multicast stream authentication scheme (Figure 1) is used only once. Hence, one can improve the security of the basic scheme without decreasing its efficiency by using unconditionally secure instead of computationally secure message authentication. However, we need to slightly modify the scheme depicted in Figure 1. First, we assume that before sending a stream, the sender selects a unique stream number N_s and includes this number in each packet including the bootstrap packet. The stream number is securely communicated to the recipients, and the recipients accept only packets with the given

stream number. Second, it is possible that given the hash of the key $H(K_i)$, a computationally unbounded adversary can compute the key K_i with high probability, and then, use the computed key to forge a packet. To prevent this, instead of using a hash of the key K_i as a commitment, the sender uses a hash of a string $S_i = K_i||r_i$, which is a concatenation of the key K_i and a random string r_i (see Figure 4). The random value r_i is secret, and its goal is to "hide" K_i from a computationally unbounded adversary by increasing the number of keys that can hash to a given value. Ideally, the adversary should not be able to learn anything about the key given the commitment $H(S_i)$. The following lemma provides a relation between the probability of successful deception of the authentication code in use and the probability of successful deception when the adversary is given a commitment to the key.

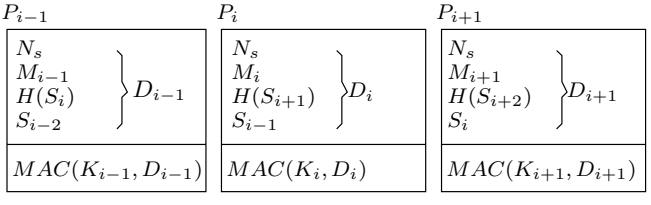

Fig. 4. A variant of the basic multicast stream authentication scheme

Lemma 1. *Let P_d be the probability of successful deception of a given authentication code, and let P_d^h be the probability of successful deception when the adversary is given a concrete commitment value $h = H(z||r)$ to the secret key z which is selected uniformly at random. Then, we have*

$$P_d^h \leq 2^{\Delta\mathcal{H}} P_d$$

where $\Delta\mathcal{H} = \mathcal{H}(z) - \mathcal{H}_\infty(z|h)$ is the difference between the entropy of the secret key and the min-entropy of the secret key given the commitment h.

The proof is given in Appendix A.

The security of the stream authentication scheme depicted in Figure 4 follows from the following theorem.

Theorem 2. *Suppose that:*

- *H is collision-resistant: it is hard to find S_i and $S_i' \neq S_i$ such that $H(S_i) = H(S_i')$,*
- *the digital signature scheme that is used to bootstrap the scheme (i.e., sign the first packet with a commitment $H(S_1)$ to the first key K_1) is unforgeable,*
- *the deception probability of the message authentication scheme given a commitment to the key is small: $\max_h P_d^h$ is negligible.*

Then, the variant of the basic scheme depicted in Figure 4 is computationally secure.

Proof. Assume that the adversary can break the stream authentication scheme. In other words, the adversary in cooperation with some of the recipients can trick another recipient u to accept a forged packet of the stream as valid. Let i be the smallest integer such that the content D'_i is accepted as valid by the recipient u when the original content D_i is different from D'_i. Since a unique sequence number is associated with each stream, if i is zero, then the adversary has managed to forge the bootstrap packet which was signed using a digital signature scheme. Let i be greater than zero. There are two possible cases: (i) the key that u used to verify the validity of D'_i is equal to the original key K_i or (ii) the key that u used to verify the validity of D'_i is different than the original key K_i. In the first case, the adversary has managed to produce a forgery for the message authentication scheme. Clearly, the probability of success in this case is negligible (not greater than $\max_h P_d^h$). In the second case, when the key K'_i used by u is different from the original key K_i, the string S'_i received by u is different from the original string S_i sent by the sender. However, the corresponding commitments $H(S_i)$ and $H(S'_i)$ must be equal since i is the smallest number such that the content D'_i is accepted as valid by the recipient u when the original content D_i sent by the sender is different from D'_i. Therefore, we can find a collision for the hash function H.

According to the previous discussion, given an adversary for the stream authentication scheme A, one can construct two adversaries: an adversary A_{ds} for the digital signature scheme that is used to sign the bootstrap packet and an adversary A_h for the hash function H that is used to commit to the keys. We can answer the signing queries of A by sending queries to the signing oracle of the digital signature scheme, selecting random keys, committing to the keys using H and using these keys to authenticate the packets of the stream. If A manages to forge the first bootstrap packet, then A_{ds} will output this forged message/signature pair. Otherwise, it will output some randomly selected message/signature pair. If A manages to produce a forgery by finding two different values S_i and S'_i that hash to the same value, then A_h will output the pair (S_i, S'_i). Otherwise, it will output some random pair. Since the probability to produce a forgery for the message authentication scheme is negligible, if A succeeds with non-negligible probability, then at least one of the adversaries A_{ds} and A_h must have a non-negligible success probability too. ■

5.2 Erasure-Tolerant Multicast Stream Authentication by Means of Cover-Free Families

In this section, we present an erasure-tolerant stream authentication scheme constructed using unconditionally secure erasure-tolerant authentication codes (η-codes).

Our scheme is derived by modifying the basic scheme depicted in Figure 4. We divide the stream into groups, each group i being a sequence of b messages $M_{i,1}, \ldots, M_{i,b}$. A unique sequence number $\langle N_s.i.j \rangle$ is assigned to each message $M_{i,j}$, where $\langle N_s.i.j \rangle$ is a concatenation of the binary representations of a unique stream number N_s, the group number i and the position of the message in the

group j. A commitment $H(S_{i+1})$ to a key string S_{i+1} and a key string S_{i-1} are included in $t+1$ packets of the group[1]. Since at most t erasures are allowed, at least one commitment copy $H(S_{i+1})$ and at least one key string copy S_{i-1} will get to the receiver. Finally, we compute and include v authentication tags in the sequence of b packets. The authentication tags are computed using an unconditionally secure authentication code as in the construction from cover-free families described in Section 4. The keys that are used to compute the tags are extracted from S_i, which is revealed in the next group of packets. If the number of tags v is large, then S_i will be a significant communication overhead. One possible solution to this problem is to use S_i as a short seed of a pseudo-random generator that will generate the secret keys of the unconditionally secure authentication codes.

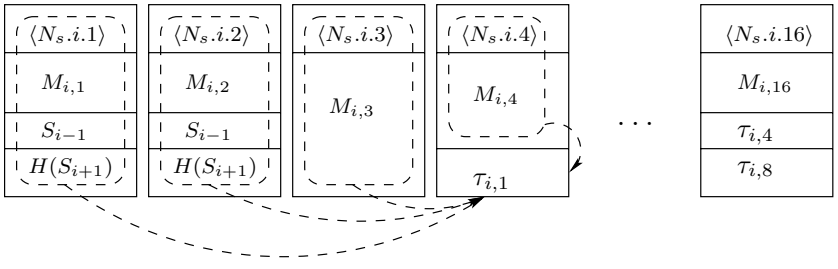

Fig. 5. Multicast stream authentication using $(1,1)$-CFF

In our simple example (Figure 5), the stream is divided into groups of 16 messages. Since only one erasure is allowed, the commitment $H(S_{i+1})$ and the seed S_{i-1} are included in only two packets. The authentication tags are computed using a $(1,1)$-cover-free family as in Figure 3. The only difference is that the tags depend on the whole content of the packets, not just on the messages $M_{i,j}$.

5.3 Erasure-Tolerant Stream Signing by Means of Cover-Free Families

To sign a lossy stream, we split the stream into groups of subsequent messages. Each group i is a sequence of b messages $M_{i,1}, \ldots, M_{i,b}$. A unique sequence number $\langle N_s.i.j \rangle$ is assigned to each message $M_{i,j}$ as in Section 5.2. The binary string $\langle N_s.i.j \rangle$ is a concatenation of the binary representations of a unique stream number N_s, the group number i and the position of the message in the group j. Finally, the groups are signed using an erasure-tolerant digital signature.

An example is given in Figure 6. In this example, the erasure-tolerant digital signature is based on the $(1,1)$-cover-free family depicted in Figure 3. For

[1] In general, we can use an erasure code to encode the commitment and the key string instead of sending multiple copies. This further optimizes the bandwidth.

example, the signature $\sigma_{i,1}$ is computed from $M_{i,1}, \ldots, M_{i,4}$ and the associated unique sequence numbers, the signature $\sigma_{i,2}$ is computed from $M_{i,5}, \ldots, M_{i,8}$ and the associated unique sequence numbers, etc.

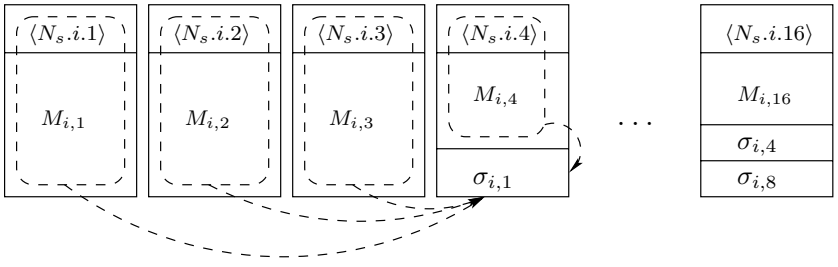

Fig. 6. Loss-tolerant stream signing scheme based on $(1, 1)$-CFF

5.4 Efficiency Issues

In the generic construction described in Section 4, multiple signatures (or authentication tags) can depend on a single message. Signing (or authenticating) each subsequence of messages anew will lead to large time complexity since each message is processed many times. Here, we present some more efficient solutions.

The Stream Authentication Case. We will use the following unconditionally secure multiple message authentication code [1]. Let $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_v$ be a sequence of v messages. The authentication tag for a message $\mathbf{a}_i$ is computed as

$$h_{x,y,z_i}(\mathbf{a}_i) = y(a_{i_0} + a_{i_1}x + \ldots + a_{i_{l-1}}x^k) + z_i = yf_{\mathbf{a}_i}(x) + z_i,$$

where $x, y, z_i, a_{i_0}, \ldots, a_{i_{l-1}} \in \mathbf{F}_q$ (q a prime power). The key parts x and y remain unchanged for all messages in the sequence. Only the part z_i is refreshed for each message.

We process each message only once and produce a sequence of b values $\alpha_1, \ldots, \alpha_b$ (see Figure 7.a). The temporary values α_i are then combined to produce the authentication tags.

The temporary values α_i are computed as

$$\alpha_i = f_{M_i}(x) = m_0 + m_1x + \ldots + m_{l-1}x^{l-1},$$

where $m_0, \ldots, m_{l-1} \in \mathbf{F}_q$ are the letters of the message M_i.

Given a subsequence $M_{i_0}, \ldots, M_{i_n}$ of messages, the authentication tag that depends on these messages can be efficiently computed as:

$$\tau = y(f_{M_{i_0}}(x) + x^l f_{M_{i_1}}(x) + \ldots + x^{nl} f_{M_{i_n}}(x)) + z_i.$$

In other words, we evaluate a polynomial over each message separately, and then combine the results to evaluate polynomials over subsequences of messages. An efficient procedure for polynomial evaluation is given in [1]. The time complexity of the procedure is 7-13 machine instructions per word.

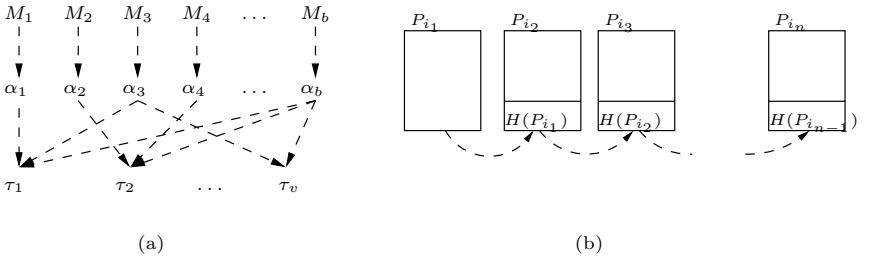

Fig. 7. Efficient computation: a) the stream authentication case, b) the stream signing case

The Stream Signing Case. In order to efficiently compute the signatures when signing a stream, we can use the hash chaining technique from [24] as depicted in Figure 7.b. Assume that we need to compute a digital signature over the subsequence of packets $P_{i_1}, \ldots, P_{i_n}$. Instead of constructing one large message consisting of the contents of the packets $P_{i_1}, \ldots, P_{i_n}$ and signing it, we can compute the hash of P_{i_1} and include it in the packet P_{i_2}, compute the hash of P_{i_2} and include it in the packet P_{i_3}, etc. We sign only the last packet P_{i_n}. Clearly, we need to process the packets only once, when we compute their hash values. Another advantage of this approach is that we can still verify the authenticity of the packets $P_{i_l}, \ldots, P_{i_n}$ even when $P_{i_{l-1}}$ is lost.

5.5 Security of the Schemes

Since a unique number $\langle N_s.i.j \rangle$ is associated with each chunk of the stream, it is trivial to show that if one is able to forge a stream signature in the scheme described in Section 5.3, then one can produce a forgery for the underlying signature scheme. A security proof of the erasure-tolerant stream authentication scheme depicted in Figure 5 can be easily obtained by extending the results presented in Section 5.1.

6 Conclusion

We provide necessary and sufficient conditions for non-degrading erasure-tolerant message authentication schemes that are based on set systems. We also present an efficient and provably secure loss-tolerant multicast stream authentication scheme that uses erasure-tolerant authentication code as a building block.

Acknowledgments

The authors thank Huaxiong Wang for providing references on covert free families.

References

1. V. Afanassiev, C. Gehrmann and B. Smeets, "Fast Message Authentication Using Efficient Polynomial Evaluation," Proceedings of Fast Software Encryption Workshop 1997, pp. 190-204.
2. R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas, and R. Needham,"A New Family of Authentication Protocols," ACM Operating Systems Review 32(4), pp. 9-20, 1998.
3. F. Bergadano, D. Cavagnino, B. Crispo, "Chained Stream Authentication," Proceeding of Selected Areas in Cryptography 2000, pp. 142-155, 2000.
4. J. Bierbrauer, T. Johansson, G. Kabatianskii and B. Smeets, "On Families of Hash Functions Via Geometric Codes and Concatenation," Proceedings of Crypto '93, pp. 331-342.
5. R. Canneti, J. Garay, G. Itkis, D. Micciancio, M. Naor and B. Pinkas, "Multicast security: A taxonomy and some efficient constructions," In Infocom '99, 1999.
6. J.L. Carter and M.N. Wegman, "Universal Classes of Hash Functions," Journal of Computer and System Sciences, Vol. 18, pp. 143-154, 1979.
7. A. Chan, "A graph-theoretical analysis of multicast authentication," Proc. of the 23rd Int. Conf. on Distributed Computing Systems, 2003.
8. S. Cheung,"An Efficient Message Authentication Scheme for Link State Routing," Proceedings of the 13th Annual Computer Security Application Conference, 1997.
9. G. Di Crescenzo, R. Graveman, R. Ge and G. Arce, "Approximate Message Authentication and Biometric Entity Authentication," Proceedings of FC 2005, LNCS 3570, pp. 240-254.
10. Y. Desmedt, Y. Frankel and M. Yung, "Multi-Receiver/Multi-Sender Network Security: Efficient Authenticated Multicast/Feedback," INFOCOM, 1992, pp.2045-2054.
11. Y. Desmedt, R. Safavi-Naini, H. Wang, L. Batten, C. Charnes and J. Pieprzyk, "Broadcast anti-jamming systems," Computer Networks Vol. 35 (2001), pp. 223-236.
12. M. Dyer, T. Fenner, A. Frieze and A. Thomson, "On key storage in secure networks," Journal of Cryptology Vol. 8 (1995), pp. 189-200.
13. K. Engel, "Interval packing and covering in the boolean lattice," Combin. Probab. Comput. Vol. 5 (1996), pp. 373-384.
14. R. Gennaro and P. Rohatgi,"How to Sign Digital Streams," Advances in Cryptology - Proceedings of Crypto '97, LNCS 1294, Springer-Verlag, 1997, pp. 180-197.
15. S. Goldwasser, S. Micali, and R. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," SIAM Journal on Computing, 17(2):281-308, April 1988.
16. R. F. Graveman, L. Xie and G. R. Arce, "Approximate Message Authentication Codes," submission to the IEEE Transactions on Image Processing, 2000.
17. G. Jakimoski, "Unconditionally Secure Information Authentication in Presence of Erasures," Proceedings of the 10th IMA International Conference on Cryptography and Coding, LNCS 3796, pp. 304 - 321, 2005.
18. W.H. Kautz and R.C. Singleton,"Nonrandom binary superimposed codes," IEEE Transactions on Information Theory Vol. 10 (1964), pp. 363-377.
19. M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical Loss-Resilient Codes," Proc. 29 th Symp. on Theory of Computing, 1997, pp. 150-159.

20. M.Luby, "LT codes," The 43rd IEEE Symposium on Foundations of Computer Science, 2002.
21. J.L. Massey, "Contemporary Cryptology: An Introduction," in Contemporary Cryptology, The Science of Information Integrity, ed. G.J. Simmons, IEEE Press, New York, 1992.
22. S. Miner and J. Staddon, "Graph-Based Authentication of Digital Streams," IEEE Symposium on Security and Privacy, 2001.
23. J.M. Park, E.K.P. Chong and H.J. Siegel, "Efficient Multicast Stream Authentication Using Erasure Codes," ACM Transactions on Information and System Security, Vol. 6, No. 2, May 2003, pp. 258-285.
24. A. Perrig, R. Canneti, J. D. Tygar, D. Song, "Efficient Authentication and Signing of Multicast Streams Over Lossy Channels," Proceedings of the IEEE Security and Privacy Symposium, 2000.
25. K.A.S. Quinn, "Bounds for key distribution patterns," Journal of Cryptology Vol. 12 (1999), pp. 227-240.
26. M. Rabin,"Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," J. ACM 36, 2, pp.335-348.
27. P.Rogaway, "Bucket hashing and its application to fast message authentication," Proceedings of CRYPTO'95, Springer-Verlag, 1995, pp. 29-42.
28. P. Rohatgi, "A compact and fast hybrid signature scheme for multicast packet authentication," In 6th ACM Conference on Computer and Communications Security, November 1999.
29. G.J. Simmons,"Authentication Theory / Coding Theory," Proceedings of Crypto '84, pp. 411-432.
30. G.J. Simmons, "A Survey of Information Authentication," in Contemporary Cryptology, The Science of Information Integrity, ed. G.J. Simmons, IEEE Press, New York, 1992.
31. R. Steinfeld, L. Bull and Y. Zheng, "Content Extraction Signatures," In the Proceedings of the International Conference on Information Security and Cryptology - ICISC 2001, LNCS 2288, p. 285, 2002.
32. D.R. Stinson, "Some Constructions and Bounds for Authentication Codes," Journal of Cryptology 1 (1988), pp. 37-51.
33. D.R. Stinson, "The Combinatorics of Authentication and Secrecy Codes," Journal of Cryptology 2 (1990), pp. 23-49.
34. D.R. Stinson, "Combinatorial Characterizations of Authentication Codes," Proceedings of Crypto '91, pp.62-73.
35. D.R. Stinson, "Universal Hashing and Authentication Codes," Proceedings of CRYPTO '91, LNCS 576 (1992), pp. 74-85.
36. D.R. Stinson, Tran van Trung and R. Wei, "Secure frameproof codes, key distribution patterns, group testing algorithms and related structures," Journal of Statistical Planning and Inference Vol. 86 (2000), pp. 595-617.
37. D.R. Stinson, R. Wei and L. Zhu, "Some new bounds for cover-free families," J. Combin. Theory A. 90 (2000), pp. 224-234.
38. P.F. Syverson, S.G. Stubblebine and D.M.Goldschlag, "Unlinkable serial transactions," In Financial Cryptography '97, Springer Verlag, LNCS 1318, 1997.
39. C. Tartary and H. Wang, "Rateless Codes for the Multicast Stream Authentication Problem," to appear in the Proceedings of IWSEC 2006.
40. M.N. Wegman and J.L. Carter, "New Hash Functions and Their Use in Authentication and Set Equality," Journal of Computer and System Sciences, Vol. 22, pp. 265-279, 1981.

41. C. K. Wong, S. S. Lam, "Digital Signatures for Flaws and Multicasts," Proceedings of IEEE ICNP '98, 1998.
42. K. Zhang,"Efficient Protocols for Signing Routing Messages," Proceedings of the Symposium on Network and Distributed System Security, 1998.

A Proof of Lemma 1

We will first show that

$$P_d^h \leq \rho P_d$$

where $\rho = \max_z \frac{P(z|h)}{P(z)}$.

We say that the message y is valid when the key is z if there is a plaintext x whose encoding under the key z is y. Let $\chi(y,z)$ be a characteristic function that takes value 1 if y is valid when the key is z, and it is 0 otherwise. Let $\phi(y_1,y_2,z)$ be a characteristic function that takes value 1 if both y_1 and y_2 are valid when the key is z, and it is 0 otherwise.

For the probability of successful impersonation, we have

$$\begin{aligned} P_I^h &= \max_{y\in\mathcal{M}} \sum_{e_z\in\mathcal{E}} \chi(y,z)P(z|h) \\ &\leq \max_{y\in\mathcal{M}} \sum_{e_z\in\mathcal{E}} \chi(y,z)\rho P(z) \\ &= \rho \max_{y\in\mathcal{M}} \sum_{e_z\in\mathcal{E}} \chi(y,z)P(z) \\ &= \rho P_I. \end{aligned}$$

For the probability of successful substitution, we have

$$\begin{aligned} P_S^h &= \sum_{y_1\in\mathcal{M}} P(y_1) \max_{y_2\in\mathcal{M}} \sum_{e_z\in\mathcal{E}} \phi(y_1,y_2,z)P(z|h) \\ &\leq \sum_{y_1\in\mathcal{M}} P(y_1) \max_{y_2\in\mathcal{M}} \sum_{e_z\in\mathcal{E}} \phi(y_1,y_2,z)\rho P(z) \\ &= \rho \sum_{y_1\in\mathcal{M}} P(y_1) \max_{y_2\in\mathcal{M}} \sum_{e_z\in\mathcal{E}} \phi(y_1,y_2,z)P(z) \\ &= \rho P_S. \end{aligned}$$

The inequality $P_d^h \leq \rho P_d$ follows from the previous two inequalities. Now, it remains to show that $\rho = 2^{\mathcal{H}(z)-\mathcal{H}_\infty(z|h)}$. Using the fact that the key z is uniformly distributed, we have

$$\begin{aligned} \log\rho &= \log \max_z \frac{P(z|h)}{P(z)} \\ &= \log \frac{\max_z P(z|h)}{P(z)} \\ &= \log \max_z P(z|h) - \log P(z) \\ &= -\mathcal{H}_\infty(z|h) + \mathcal{H}(z). \end{aligned}$$

■

A Practical and Tightly Secure Signature Scheme Without Hash Function*

Benoît Chevallier-Mames[1,**] and Marc Joye[2]

[1] Gemalto, Security Labs
Avenue du Jujubier, 13705 La Ciotat Cedex, France
benoit.chevallier-mames@gemalto.com
[2] Thomson R&D France
Technology Group, Corporate Research, Security Laboratory
1 avenue de Belle Fontaine, 35576 Cesson-Sévigné, France
marc.joye@thomson.net

Abstract. In 1999, two signature schemes based on the flexible RSA problem (a.k.a. strong RSA problem) were independently introduced: the Gennaro-Halevi-Rabin (GHR) signature scheme and the Cramer-Shoup (CS) signature scheme. Remarkably, these schemes meet the highest security notion in the *standard model*. They however differ in their implementation. The CS scheme and its subsequent variants and extensions proposed so far feature a loose security reduction, which, in turn, implies larger security parameters. The security of the GHR scheme and of its twinning-based variant are shown to be tightly based on the flexible RSA problem but additionally (i) either assumes the existence of *division-intractable* hash functions, or (ii) requires an *injective* mapping into the prime numbers in both the signing *and* verification algorithms.

In this paper, we revisit the GHR signature scheme and completely remove the extra assumption made on the hash functions without relying on injective prime mappings. As a result, we obtain a *practical* signature scheme (and an on-line/off-line variant thereof) whose security is *solely* and *tightly* related to the strong RSA assumption.

Keywords: Digital signatures, standard model, strong RSA assumption, tight reduction, Gennaro-Halevi-Rabin signature scheme, Cramer-Shoup signature scheme, on-line/off-line signatures.

1 Introduction

Digital signatures are one of the most useful and fundamental primitives resulting from the invention of public-key cryptography by Diffie and Hellman [DH76] in 1976. Rivest, Shamir and Adleman [RSA78] gave the first practical implementation of such a primitive. However, at that time, the security analysis of

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

** Also with École Normale Supérieure, Département d'Informatique, 45 rue d'Ulm, 75230 Paris 05, France.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 339–356, 2007.

signature schemes was studied more heuristically: a scheme was declared "secure" if no attacks were found.

Provably signature schemes. Formal security notions for signature schemes were later introduced by Goldwasser, Micali and Rivest in their seminal paper [GMR88]. They also proposed a signature scheme provably meeting their security notion (see also [Gol86, NY89]). This tree-based signature scheme was subsequently improved by Dwork and Naor [DN94], Cramer and Damgård [CD96], and Catalano and Gennaro [CG05].

Random oracle model. More efficient schemes were proven secure in the so-called random oracle model [BR93, FS87]. The random oracle model assumes that the output of a hash function behaves like a random generator. Provably secure signature schemes relying on this extra assumption are presented and discussed in [BR96, PS96, GJ03, KW03, BLS04], with different underlying problems: the discrete logarithm problem [PS96], the RSA problem [BR96, KW03], the CDH problem [GJ03, KW03], the DDH problem [KW03], or the CDH problem on certain elliptic curves [BLS04]. The idealized random oracle model has however certain limitations [CGH98, PV05].

Standard model. Efficient signature schemes without random oracles are due to Gennaro, Halevi and Rabin [GHR99] and to Cramer and Shoup [CS00]. They are both based on the strong RSA assumption, which assumes that it is impossible to find an e-th modular root of a given element, even if e can be chosen arbitrarily by the attacker (provided of course that $e \geq 2$). Subsequent improvements and modifications include the works of [NPS01, Zhu01, CL02, Fis03], with some better performances and signature size, or additional features for particular use cases.

More recently, the introduction of cryptographic bilinear mappings has allowed the emergence of new techniques to achieve security without random oracles. More precisely, the study of pairings gave rise to signature scheme based on the strong Diffie-Hellman assumption [BB04] and even more recently on the computational Diffie-Hellman assumption [Wat05, BSW06].

Our Contribution. This paper presents a new signature scheme based on the strong RSA assumption, in the standard model. In contrast to the Cramer-Shoup scheme and its variants, our security proof yields a tight reduction. Moreover, our scheme does not rely on special-type hash functions nor injective prime functions. In this sense, it is easier to implement than the Gennaro-Halevi-Rabin scheme and its known variants, as one needs not to design such functions. Finally, our scheme features an efficient on-line/off-line variant.

Organization. The rest of this paper is organized as follows. In Section 2, we introduce some background on signature schemes, provable security and RSA-related problems. In Section 3, we briefly review the Gennaro-Halevi-Rabin and the Cramer-Shoup signature schemes. Section 4 is the main part of our paper. We introduce our new signature scheme (that we call TSS) and prove its security in the standard model. We also compare our scheme with prior RSA-based

schemes in the standard model, and present an on-line/off-line variant. Finally, we conclude in Section 5.

2 Preliminaries

In this section, we introduce notations and definitions that are used throughout the paper. For convenience, we often identify an integer with its binary representation: $a \in \{0,1\}^\ell$ is also viewed as an integer in the range $[0, 2^\ell - 1]$. We say that a is an ℓ-bit integer if a is an integer in the range $[2^{\ell-1}, 2^\ell - 1]$. An (odd) prime p is a strong prime if $(p-1)/2$ is prime. An RSA modulus $n = pq$ is safe if it is the product of two equal-size strong primes.

2.1 Signature Schemes

A signature scheme $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is defined by the three following algorithms:

1. *Key generation algorithm* Gen. On input security parameter k, algorithm Gen produces a pair $(\mathsf{pk}, \mathsf{sk})$ of matching public and private keys.
2. *Signing algorithm* Sign. Given a message m in a set $\mathcal{M}$ of messages and a pair of matching public and private keys $(\mathsf{pk}, \mathsf{sk})$, Sign produces a signature σ. The signing algorithm can be probabilistic.
3. *Verification algorithm* Verify. Given a signature σ, a message $m \in \mathcal{M}$ and a public key pk, Verify checks whether σ is a valid signature on m with respect to pk.

Several security notions have been defined for signature schemes, mostly based on the work by Goldwasser, Micali and Rivest [GMR88]. It is now customary to ask for the infeasibility of existential forgeries, even against adaptive chosen-message adversaries:

- An *existential forgery* is a signature on a new message, valid and generated by the adversary. The corresponding security goal is called *existential unforgeability* (EUF).
- A *weak existential forgery* is a new message/signature pair, valid and generated by the adversary. The corresponding security goal is called *strong existential unforgeability* (sEUF).
- The verification key is public to anyone, including to the adversary. But more information may also be available. The strongest kind of attack scenario is formalized by the *adaptive chosen-message attacks* (CMA), where the adversary can ask the signer to sign any message of her choice, in an adaptive way.

As a consequence, we say that a signature scheme is *secure* if it prevents (weak) existential forgeries against chosen-message attacks (EUF-CMA or sEUF-CMA) with overwhelming probability.

– A signature scheme Sig is (τ, q_s, ε)-*secure* if the success probability

$$\mathsf{Succ}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig}}(\mathcal{A}, q_s) := \Pr\left[\begin{array}{c}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^k), (m_*, \sigma_*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk};\cdot)}(\mathsf{pk}) : \\ \mathsf{Verify}(\mathsf{pk}; m_*, \sigma_*) = \text{true}\end{array}\right] < \varepsilon$$

for any adversary $\mathcal{A}$ with running time bounded by τ, making (at most) q_s queries to a signing oracle $\mathsf{Sign}(\mathsf{sk};\cdot)$, and returning a valid signature σ_* on a message m_* that was not submitted to the signing oracle.

– A signature scheme Sig is *strongly* (τ, q_s, ε)-*secure* if the success probability

$$\mathsf{Succ}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sig}}(\mathcal{A}, q_s) := \Pr\left[\begin{array}{c}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^k), (m_*, \sigma_*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk};\cdot)}(\mathsf{pk}) : \\ \mathsf{Verify}(\mathsf{pk}; m_*, \sigma_*) = \text{true}\end{array}\right] < \varepsilon$$

for any adversary $\mathcal{A}$ with running time bounded by τ, making (at most) q_s queries to a signing oracle $\mathsf{Sign}(\mathsf{sk};\cdot)$, and returning a valid message/signature pair (m_*, σ_*) where m_* was not submitted to the signing oracle and/or σ_* was not returned by the signing oracle.

2.2 Provable Security

Reduction. Basically, the idea behind provable security (or reductionist security [KM04]) is to prove that a scheme is secure by exhibiting a so-called reduction that uses a chosen-message attacker against the signature scheme, in order to solve a hard cryptographic problem. In the *standard model*, the attacker has access to a signing oracle that is simulated by the reduction, answering the q_s signature queries on chosen messages, and receiving eventually a signature forgery.

Tightness of reduction. Two classes of provably secure signature schemes can be distinguished. The first class proposes reductions that are said *loose*, as they can turn an attacker into an algorithm solving a cryptographic problem, but whose ratio

$$\rho = \frac{\text{running time}}{\text{success probability}}$$

is far greater than those required for the attacker to produce a forgery. Hence, this kind of reduction only proves the security asymptotically. The second class of provable signature schemes features so-called *tight* reductions, using the attacker to solve the cryptographic problem with roughly the same ratio ρ.

Signature schemes in the standard model. Of course, secure schemes with a reduction in the standard model are the preferred ones, even if proofs in the random oracle model are arguments for a good design. Indeed, important differences between the idealized random oracle model and the real life have been pointed out in the literature [CGH98, PV05].

There are just a handful of practical signature schemes with a security reduction in the standard model, and most of them rely on *flexible* problems,

one-more problems or *q-type* problems. A notable exception is the recent signature scheme of Waters [Wat05] (see also [BSW06]), the security of which relies on the computational Diffie-Hellman [DH76] assumption, in bilinear groups.

Flexible problems are cryptographic problems admitting several solutions, the solver has just to find one of them. Its main representative is certainly the flexible RSA problem [BP97, FO97].

Definition 1 (Flexible RSA problem – SRSA). *Being given a safe RSA modulus n and an element $y \in \mathbb{Z}_n^*$, the* flexible RSA problem *is defined as finding an element $x \in \mathbb{Z}_n^*$ and an integer $e > 1$ such that $x^e \equiv y \pmod{n}$.*

The strong RSA assumption *conjectures that there is no attacker that can (τ, ε)-solve the flexible* RSA *problem (i.e., with success probability smaller than ε and running time bounded by τ) with τ polynomial and ε non-negligible.*

One-more problems are problems where the solver has n accesses to an oracle that solves a hard problem, in order to solve $n+1$ instances of this hard problem. Typical examples include the one-more RSA, the one-more DL and the one-more CDH [BNPS03].

Finally, q-type problems are problems where the solver receives a large instance of q data, and must find a value satisfying a certain relation with this instance. Notably, the returned value could be a (new) data of the same kind. A typical example of this last type of problem is the strong Diffie-Hellman problem [BB04].

Devising an efficient signature scheme based on an harder type of problem such as the RSA problem [RSA78], the discrete logarithm problem or the computational Diffie-Hellman problem — *in a group without pairing* — in the standard model still remains an open question.

3 Signature Schemes Based on the Strong-RSA Assumption

We briefly review several *efficient* signature schemes the security of which relies on the strong RSA assumption, in the standard model. We refer the reader to the original papers for a complete description.

We begin with the two main schemes introduced in 1999: the Gennaro-Halevi-Rabin (GHR) signature scheme [GHR99] and the Cramer-Shoup (CS) signature scheme [CS00]. The CS scheme was subsequently modified in several directions, including the variants by Zhu [Zhu01, Zhu03], Camenisch and Lysyanskaya [CL02] and Fischlin [Fis03]. As the Twin-GHR scheme [NPS01], our new scheme (see Section 4) as for it can be viewed as a variant of the GHR scheme.

In order to allow an easier comparison between the different schemes, we deviate from the original papers and use similar notation when describing the signature schemes. For an RSA modulus n, we let $\mathrm{QR}_n \subset \mathbb{Z}_n^*$ denote the set of quadratic residues modulo n. The message space is denoted by $\mathcal{M}$; $\mathcal{H}$ denotes a hash function.

3.1 Gennaro-Halevi-Rabin Signature Scheme

The message space is $\mathcal{M} = \{0,1\}^*$ and $\mathcal{H} : \mathcal{M} \to \{0,1\}^{\ell_h}$ is division-intractable.

Key generation: The public key is $\mathsf{pk} = \{n, u\}$ where $n = (2p'+1)(2q'+1)$ is a safe RSA modulus and u is a random element in $\mathbb{Z}_n^*$. The private key is $\mathsf{sk} = \{p', q'\}$.

Signing: The signature on a message $m \in \mathcal{M}$ is given by $\sigma = u^{c^{-1} \bmod 2p'q'} \bmod n$ where $c = \mathcal{H}(m)$.

Verification: Signature σ on message $m \in \mathcal{M}$ is accepted iff $\sigma^{\mathcal{H}(m)} \equiv u \pmod n$.

In the original description, hash function $\mathcal{H}$ has to be a *division-intractable* hash function (see [GHR99] for a precise definition). Following [CN00], the easiest way to achieve this additional property is to define $\mathcal{H}$ as a hash function that maps bitstrings to prime numbers.[1]

Tight variant. The GHR signature scheme as above has a *loose* security reduction to the flexible-RSA problem [GHR99, Cor00]. However, in [GHR99], the authors also propose techniques to achieve tightness in their signature scheme.

Basically, their idea is to make use of a chameleon hash function [KR00]. Let P be an ℓ_p-bit prime, let Q be an ℓ_q-bit prime divisor of $P-1$, and let $\langle g \rangle$ denote the cyclic subgroup generated by an element $g \in \mathbb{Z}_P^*$ of order Q. They so obtain a scheme such that an attacker against it can be used to solve either the discrete logarithm problem in subgroup $\langle g \rangle$ or the flexible RSA problem modulo n, with roughly the same success probability and running time.

The message space is $\mathcal{M} = \mathbb{Z}_Q$ and $\mathcal{H} : \langle g \rangle \to \{0,1\}^{\ell_h}$. The scheme then becomes:

Key generation: The public key is $\mathsf{pk} = \{n, u, g, y, P\}$ where $n = (2p'+1)(2q'+1)$ is a safe RSA modulus, u is a random element in $\mathbb{Z}_n^*$ and y is a random element in $\langle g \rangle \subseteq \mathbb{Z}_P^*$. The private key is $\mathsf{sk} = \{p', q'\}$.

Signing: The signature on a message $m \in \mathcal{M}$ is given by $\sigma = (r, s)$ where r is a random element in $\mathbb{Z}_Q$ and $s = u^{c^{-1} \bmod 2p'q'} \bmod n$ with $c = \mathcal{H}(g^m y^r \bmod P)$.

Verification: Signature $\sigma = (r, s)$ on message $m \in \mathcal{M}$ is accepted iff $s^{c'} \equiv u \pmod n$ with $c' = \mathcal{H}(g^m\, y^r \bmod P)$.

Clearly, the chameleon hash function could be of different nature: for a joint use with GHR scheme, most interesting cases are certainly chameleon hash functions based on RSA or factorization problems (*e.g.*, [KS06]).

[1] It would also be possible to remove the hash function, and to sign instead only prime messages. However, this solution suffers from practicality.

Twinning-based variant. The twinning paradigm was introduced by Naccache, Pointcheval and Stern in [NPS01]. It particulary fits to GHR signatures.

Let $\mathcal{P}$ be an injective function that maps the set $\{0,1\}^{2\ell_m}$ into the prime numbers. Consider for example the function

$$\mathcal{P} : \{0,1\}^{2\ell_m} \rightarrow \{\text{primes}\},\ \mu \mapsto \texttt{NextPrime}(\mu\, 2^{\tau})$$

with $\tau \approx 5\log_2(2\ell_m)$, which can be evaluated with less than $40\,\ell_m$ primality tests [NPS01, Appendix B]. The Twin-GHR scheme is then defined as follows:

Key generation: The public key is $\mathsf{pk} = \{n, N, u_1, u_2\}$ where $n = (2p'+1)(2q'+1)$ and $N = (2P'+1)(2Q'+1)$ are two safe RSA moduli, u_1 is a random element in $\mathbb{Z}_n^*$ and u_2 is a random element in $\mathbb{Z}_N^*$. The private key is $\mathsf{sk} = \{p', q', P', Q'\}$.

Signing: The signature on a message $m \in \mathcal{M} = \{0,1\}^{\ell_m}$ is given by $\sigma = (\mu_1, s_1, s_2)$, computed, for a random $\mu_1 \in \{0,1\}^{2\ell_m}$ and $\mu_2 = (m\|m) \oplus \mu_1$, as $s_1 = {u_1}^{\mathcal{P}(\mu_1)^{-1} \bmod 2p'q'} \bmod n$ and $s_2 = {u_2}^{\mathcal{P}(\mu_2)^{-1} \bmod 2P'Q'} \bmod N$.

Verification: Signature $\sigma = (\mu_1, s_1, s_2)$ on message $m \in \mathcal{M}$ is accepted iff $s_1^{\mathcal{P}(\mu_1)} = u_1 \bmod n$ and $s_2^{\mathcal{P}((m\|m)\oplus\mu_1)} = u_2 \bmod N$.

The Twin-GHR signature scheme has a *tight* security reduction to the flexible-RSA problem [NPS01].

3.2 Cramer-Shoup Signature Scheme

The message space is $\mathcal{M} = \{0,1\}^*$ and $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^{\ell_h}$.

Key generation: The public key is $\mathsf{pk} = \{n, e, x, h\}$ where $n = (2p'+1)(2q'+1)$ is a safe RSA modulus, e is an (ℓ_h+1)-bit prime and x, h are random elements in QR_n. The private key is $\mathsf{sk} = \{p', q'\}$.

Signing: The signature on a message $m \in \mathcal{M}$ is given by $\sigma = (c, u, v)$ where c is a random (ℓ_h+1)-bit prime, u is a random element in QR_n and $v = (x\, h^{\mathcal{H}(w)})^{c^{-1} \bmod p'q'} \bmod n$ with $w = u^e h^{-\mathcal{H}(m)} \bmod n$.

Verification: Signature $\sigma = (c, u, v)$ on message $m \in \mathcal{M}$ is accepted iff (1) c is an odd (ℓ_h+1)-bit integer and (2) $v^c\, h^{-\mathcal{H}(w')} \equiv x \pmod n$ with $w' = u^e\, h^{-\mathcal{H}(m)} \bmod n$.

This signature scheme, as shown in [CS00], is secure under the strong RSA assumption. We refer the reader to the original paper for details, and just mention that unfortunately, the reduction is loose, with a loss factor equal to $\frac{1}{q_s}$, where q_s is the number of signature queries the attacker is allowed to make.[2] On

[2] To be complete, the reduction of CS scheme is made of two types of reductions, one of which tightly reduces the security to the flexible RSA problem, while the other one reduces the security to the (plain) RSA problem — but with a $\frac{1}{q_s}$ factor. Even if flexible RSA should be easier than RSA, there is no estimation of the difference of difficulty between these problems. Note also that a loose reduction implies the use of larger RSA moduli.

the other hand, one of the advantages of the CS scheme compared to the GHR scheme is that the hash function $\mathcal{H}$ needs not to map to prime numbers nor to be division-intractable, but merely to be collision-resistant.

Camenisch-Lysyanskaya signature scheme. Camenisch and Lysyanskaya introduce in [CL02] a variant of CS signature scheme (that we abbreviate in CL). Independently, in a Chinese journal [Zhu01], Zhu propose a similar scheme (see also [Zhu03]).

Key generation: The public key is $\mathsf{pk} = \{n, x, g, h\}$ where $n = (2p'+1)(2q'+1)$ is a safe RSA modulus, prime and x, g, h are random elements in QR_n. The private key is $\mathsf{sk} = \{p', q'\}$.

Signing: The signature on a message $m \in \mathcal{M} = \{0,1\}^{\ell_m}$ is given by $\sigma = (c, t, v)$ where c is a random ℓ_c-bit prime with $\ell_c \geq (\ell_m + 2)$, t is a random $(\ell_n + \ell_m + \ell)$-bit integer and $v = (x\, g^t\, h^m)^{c^{-1} \bmod p'q'} \bmod n$.

Verification: Signature $\sigma = (c, t, v)$ on message $m \in \mathcal{M}$ is accepted iff (1) c is an odd ℓ_c-bit integer and (2) $v^c\, g^{-t}\, h^{-m} \equiv x \pmod n$.

Fischlin signature scheme. A last variant is due to Fischlin in [Fis03]. The message space is $\mathcal{M} = \{0,1\}^*$ and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{\ell_h}$.

Key generation: The public key is $\mathsf{pk} = \{n, x, g, h\}$ where $n = (2p'+1)(2q'+1)$ is a safe RSA modulus, prime and x, g, h are random elements in QR_n. The private key is $\mathsf{sk} = \{p', q'\}$.

Signing: The signature on a message $m \in \mathcal{M}$ is given by $\sigma = (c, t, v)$ where c is a random $(\ell_h + 1)$-bit prime, t is a random ℓ_h-bit integer and $v = (x\, g^t\, h^{t \oplus \mathcal{H}(m)})^{c^{-1} \bmod p'q'} \bmod n$.

Verification: Signature $\sigma = (c, t, v)$ on message $m \in \mathcal{M}$ is accepted iff (1) c is an odd $(\ell_h + 1)$-bit integer and (2) $v^c\, g^{-t} h^{-(t \oplus \mathcal{H}(m))} \equiv x \pmod n$.

A comparison of all the previous schemes is presented in Table 1, Section 4.3.

4 The TSS Signature Scheme

This section is the core of our paper. We introduce the TSS (for *Tightly Secure Signature*) scheme and prove its security in the standard model. Our reduction is tightly related to the flexible RSA problem.

The GHR scheme requires the use of a hash function that maps to prime numbers (or at least, that is division-intractable). The CS scheme and its variants, on the other hand, feature a loose security reduction: there is a loss factor of $1/q_s$. Our goal, when designing TSS, is to combine the practicality of the CS scheme with the tightness of the GHR scheme.

4.1 Description

Intuitively, there are two ideas behind our scheme. First, as in the CS-like schemes (cf. Section 3), we want to use a fresh, prime exponent c in each signature generation and then to give a c^{th} root modulo a safe RSA modulus n as part of the signature. Further, we want prime c to be free of any particular relation (except its size), in order to allow the use of fast prime generation algorithms. The second idea is, as in the GHR scheme, to use a chameleon function in order to tighten the security reduction. In order to base the security on the strong RSA assumption, we define a second safe RSA modulus N and make use of an RSA-type chameleon function.

The message space is $\mathcal{M} = \{0,1\}^{\ell_m}$. Let also ℓ_n be a security parameter. A detailed description of the TSS scheme is given below.

Gen: On input ℓ_n and ℓ_m:
- choose an (odd) $(\ell_m + 1)$-bit prime E;
- generate two random ℓ_n-bit safe RSA moduli $n = (2p' + 1)(2q' + 1)$ and $N = (2P' + 1)(2Q' + 1)$ such that $\gcd(P'Q', E) = 1$;
- compute $D = E^{-1} \bmod 2P'Q'$;
- choose at random two elements $u \in \mathbb{Z}_n^*$ and $g \in \mathbb{Z}_N^*$.

The public key is $\mathsf{pk} = \{N, n, u, g, E\}$ and the private key is $\mathsf{sk} = \{p', q', D\}$.

Sign: Let $m \in \{0,1\}^{\ell_m}$ be the message to be signed:
- choose a random prime c in $[(N+1)/2, N[$;
- compute $s = u^{c^{-1} \bmod 2p'q'} \bmod n$ and $r = (c\, g^{-(m+1)})^D \bmod N$.

The signature on m is $\sigma = (r, s) \in \mathbb{Z}_N^* \times \mathbb{Z}_n^*$.

Verify: Let $\sigma = (r, s)$ be a putative signature on message $m \in \{0,1\}^{\ell_m}$. Then:
(i) check that $(r, s) \in [0, N[\times [0, n[$;
(ii) check that $s^c \equiv u \pmod n$ where $c = g^{m+1}\, r^E \bmod N$.
If the two conditions hold then signature σ is accepted.

Remark 1. If Condition (i) is removed in the verification phase, *i.e.*, $(r, s) \stackrel{?}{\in} [0, N[\times [0, n[$, the corresponding security level becomes EUF-CMA. It is easy to see that the scheme is no longer sEUF-CMA because if (r, s) is a valid signature on a message m then so is $(r + r_0\, N, s + s_0\, n)$ on the same message m, for arbitrary integers r_0 and s_0.

Remark 2. As in the CS scheme, there is no need to check the primality of c in the verification algorithm. In the TSS scheme, the verification step is even simpler, as one needs to verify neither the parity nor the bitsize of c.

Typically, we set $\ell_m = 160$ and $\ell_n = 1024$, for using TSS with short messages, but there is actually no limitation on ℓ_m: if signing long messages is required, one could set ℓ_m to a large value,[3] and achieve security under the sole strong-RSA

[3] Actually this is basically as in the Camenisch-Lysyanskaya signature scheme [CL02].

assumption. Alternatively, it is also possible to use TSS preceded by a hashing step of the message, at the price of assuming in addition the collision resistance of the underlying hash function.

The previous scheme is subject to numerous variants. One can for example (slightly) speed up the signature algorithm by pre-computing $g^{-1} \bmod N$ or $g^{-D} \bmod N$ and then by evaluating r as $r = (c\,(g^{-1})^{m+1})^D \bmod N$ or as $r = c^D\,(g^{-D})^{m+1} \bmod N$.

4.2 Security Analysis

We show that the security of our scheme is *tightly* related the strong RSA assumption. That is, given an ℓ_n-bit safe RSA modulus $\hat{n}$ and a random element $\hat{y} \in \mathbb{Z}_{\hat{n}}^*$, we want to find a pair $(\hat{x}, \hat{e}) \in \mathbb{Z}_{\hat{n}}^* \times \mathbb{Z}_{>1}$ satisfying $\hat{y} \equiv \hat{x}^{\hat{e}} \pmod{\hat{n}}$. More formally, we prove the following theorem.

Theorem 1. *Suppose that the flexible* RSA *problem is (τ, ε)-hard. Then, for any q_s, the* TSS *signature scheme is strongly $(\tau_{\mathcal{A}}, q_s, \varepsilon_{\mathcal{A}})$-secure, where*

$$\varepsilon \geq \frac{\varepsilon_{\mathcal{A}}}{2} \quad \textit{and} \quad \tau \lesssim \tau_{\mathcal{A}} + \mathrm{O}\big({\ell_n}^5 + q_s\,{\ell_n}^3 \max(\log q_s, \ell_n)\big)\ .$$

Proof. As usual, the proof is by contradiction. We assume that there exists a polynomial-time adversary $\mathcal{A}$ that is able to produce a weak existential forgery with non-negligible success probability $\varepsilon_{\mathcal{A}}$ within time $\tau_{\mathcal{A}}$ after q_s queries to a signing oracle. We then use $\mathcal{A}$ to (τ, ε)-solve the flexible RSA problem, *i.e.*, to find a pair $(\hat{x}, \hat{e})$ on input challenge $(\hat{n}, \hat{y})$.

– We toss a coin $b \in \{0, 1\}$ and run Simulation b defined as follows.

Simulation 0

- We let $n = \hat{n}$. We choose an (odd) $(\ell_m + 1)$-bit prime E. Next, we generate a random ℓ_n-bit safe RSA modulus $N = (2P' + 1)(2Q' + 1)$ such that $\gcd(P'Q', E) = 1$. We compute $D = E^{-1} \bmod 2P'Q'$. We choose a random element $g \in \mathbb{Z}_N^*$. Finally, for all $i \in \{1, \ldots, q_s\}$, we let c_i be a random prime in $[(N + 1)/2, N[$ and define

$$u = \hat{y}^{\prod_i c_i} \bmod n\ .$$

  We create the public key $\mathsf{pk} = \{N, n, u, g, E\}$. It is easy to see that the key generation is perfectly simulated.

- When $\mathcal{A}$ requests the signature on a message $m_j \in \{0, 1\}^{\ell_m}$, for $j \in \{1, \ldots, q_s\}$, we simulate the signing oracle by computing

$$r_j = (c_j\, g^{-(m_j+1)})^D \bmod N \quad \text{and} \quad s_j = \hat{y}^{\prod_{i \neq j} c_i} \bmod n\ .$$

  We return $\sigma_j = (r_j, s_j)$ as the signature on m_j. Here too, the simulation is perfect.

Simulation 1

- We let $N = \hat{n}$ and $g = \hat{y}$. We choose a random ℓ_n-bit safe RSA modulus $n = (2p' + 1)(2q' + 1)$ and an (odd) $(\ell_m + 1)$-bit prime E. (W.l.o.g., we may assume that (odd) prime $E \in \mathbb{Z}^*_{2P'Q'}$ as otherwise we would have $E = P'$ or $E = Q'$, which yields the factorization of N.) Finally, we choose a random element $u \in \mathbb{Z}^*_n$.
  We create the public key $\mathsf{pk} = \{N, n, u, g, E\}$. The key generation is perfectly simulated.
- When $\mathcal{A}$ requests the signature on a message $m_j \in \{0,1\}^{\ell_m}$, for $j \in \{1, \ldots, q_s\}$, we simulate the signing oracle as follows.
  1. We choose a random element $r_j \in \mathbb{Z}^*_N$ and define $c_j = g^{m_j+1} {r_j}^E \bmod N$;
  2. If c_j is not a prime lying in $[(N+1)/2, N[$, then we go back to Step 1.

  Next, we compute $s_j = u^{{c_j}^{-1} \bmod 2p'q'} \bmod n$ and return $\sigma_j = (r_j, s_j)$ as the signature on m_j. The simulation is perfect.

– Eventually, adversary $\mathcal{A}$ outputs with probability $\varepsilon_{\mathcal{A}}$ a valid signature forgery $\sigma_* = (r_*, s_*) \in [0, N[\times [0, n[$ on a message $m_* \in \{0,1\}^{\ell_m}$, with $(m_*, \sigma_*) \neq (m_i, \sigma_i)$ for all $i \in \{1, \ldots, q_s\}$. We compute $c_* := g^{m_*+1} {r_*}^E \bmod N$.

- If $c_* \neq c_j$ for all $j \in \{1, \ldots, q_s\}$, if $c_* > 1$, and if $b = 0$ (*i.e.*, Simulation 0 was run) then it follows that $\gcd(c_*, \prod_i c_i) = 1$, since $c_* \in [2, N[$ and all c_i's are primes in set $[(N+1)/2, N[$. Hence, from extended Euclidean algorithm, we get integers α and β s.t. $\alpha\, c_* + \beta \prod_i c_i = 1$. Therefore, noting that $\hat{y}^{\prod_i c_i} \equiv u \equiv {s_*}^{c_*} \pmod{n}$ and $n = \hat{n}$, we have

$$\hat{y} \equiv \hat{y}^{\alpha\, c_* + \beta \prod_i c_i} \equiv \left(\hat{y}^{\alpha}\, {s_*}^{\beta}\right)^{c_*} \pmod{\hat{n}} \ .$$

  The pair $(\hat{x}, \hat{e})$ with $\hat{x} := \hat{y}^{\alpha}\, {s_*}^{\beta} \bmod \hat{n}$ and $\hat{e} := c_*$ is thus a solution to the flexible RSA problem.

- If $c_* = c_j$ for some $j \in \{1, \ldots, q_s\}$ (and thus $s_* = s_j$) and if $b = 1$ (*i.e.*, Simulation 1 was run) then, remembering that $N = \hat{n}$ and $g = \hat{y}$, we get

$$\begin{cases} g^{m_j+1} {r_j}^E \equiv g^{m_*+1} {r_*}^E \pmod{N} \implies \hat{y}^{(m_j - m_*)} \equiv \left(\dfrac{r_*}{r_j}\right)^E \pmod{\hat{n}}\,, \\ s_j = s_* \ . \end{cases}$$

  Note that we cannot have $m_* = m_j$ as otherwise we would have $r_* = r_j$ and so $(m_*, \sigma_*) = (m_j, \sigma_j)$, a contradiction. Therefore, since E is an $(\ell_m + 1)$-bit integer, we can find integers α and β by the extended Euclidean algorithm so that $\alpha\, E + \beta\, (m_j - m_*) = \gcd(E, m_j - m_*) = 1$. As a result, we have

$$\hat{y} \equiv \hat{y}^{\alpha\, E + \beta\, (m_j - m_*)} \equiv \left(\hat{y}^{\alpha}\, (r_*/r_j)^{\beta}\right)^E \pmod{\hat{n}}$$

  and the pair $(\hat{x}, \hat{e})$ with $\hat{x} := \hat{y}^{\alpha}\, (r_*/r_j)^{\beta} \bmod \hat{n}$ and $\hat{e} = E$ is a solution to the flexible RSA problem.

- If $c_* = 0$ and if $b = 0$ (*i.e.*, Simulation 0 was run) then, letting $\Lambda = \prod_i c_i$, we compute $\hat{d} := \hat{e}^{-1} \bmod \Lambda$ for an arbitrary $\hat{e} > 1$ such that $\gcd(\hat{e}, \Lambda) = 1$. So, the pair $(\hat{x}, \hat{e})$ with $\hat{x} := \hat{y}^{\hat{d}} \bmod \hat{n}$ is a solution to the flexible RSA problem:
$$\hat{x}^{\hat{e}} \equiv \hat{y}^{\hat{e}\hat{d}} \equiv \hat{y}^{\hat{e}\hat{d} \bmod \Lambda} \equiv \hat{y} \pmod{n}$$
because $c_* = 0$ implies $u = 1$ and thus $\hat{y}^{\Lambda} \bmod \hat{n} = 1$ (remember that $n = \hat{n}$ when $b = 0$).
- If $c_* = 1$ and if $b = 1$ (*i.e.*, Simulation 1 was run) then, using extended Euclidean algorithm, we can find integers α and β s.t $\alpha\, E + \beta\, (m_* + 1) = \gcd(E, m_* + 1) = 1$.[4] Hence, since $c_* = 1 = \hat{y}^{m_*+1}\, {r_*}^E \bmod N$, we get
$$\hat{y} \equiv \hat{y}^{\alpha\, E + \beta\, (m_*+1)} \equiv \left(\hat{y}^{\alpha}\, {r_*}^{-\beta}\right)^E \pmod{\hat{n}} \ .$$
Consequently, the pair $(\hat{x}, \hat{e})$ with $\hat{x} := \hat{y}^{\alpha}\, {r_*}^{-\beta} \bmod \hat{n}$ and $\hat{e} = E$ is a solution to the flexible RSA problem.

Since $\mathcal{A}$'s view is perfectly simulated, the success probability of the reduction is clearly $\varepsilon_{\mathcal{A}}/2$.

For Simulation 0, we need to generate ℓ_n-bit safe RSA modulus N, $(\ell_m + 1)$-bit prime E, ℓ_n-bit modular inverse D and ℓ_n-bit parameter u in the key generation; we also need, for each signature query, compute r_j and s_j. We assume that we have algorithms so that the generation of safe prime is quintic, the generation of a prime is quartic and the evaluation of a modular exponentiation or of a modular inverse is cubic, in the bitlength. The evaluation of u and the q_s s_j's amounts to $\mathrm{O}(q_s \log q_s)$ ℓ_n-bit exponentiations using the trick of [CLP05, § 3.3]. Hence, the running time required by the reduction is (approximatively) $\tau_{\mathcal{A}} + \mathrm{O}({\ell_n}^5 + q_s \log q_s\ {\ell_n}^3)$.

For Simulation 1, further assuming that primality testing is cubic in the bitlength, we similarly obtain that the running time required by the reduction is (approximatively) $\tau_{\mathcal{A}} + \mathrm{O}({\ell_n}^5 + q_s\, {\ell_n}^4)$. □

4.3 Comparison with Other Schemes

In Table 1, we compare the advantages and drawbacks of the schemes presented in Section 3 with our TSS scheme, including the differences in tightness of security reduction in the standard model, the size of signatures and the size of public/private keys. When applicable, we also give necessary conditions the hash function should fulfill (in addition to collision resistance).

From this table, it appears that the TSS scheme is proven secure *solely* under the strong-RSA assumption, with a *tight* security reduction. Furthermore, this is not done at the price of extra properties on a hash function, as the division-intractability for the GHR scheme. Twin-GHR is also tightly and solely related to the strong RSA assumption. Twin-GHR and TSS however differ in their

[4] This last case explains why $(m + 1)$ (and not merely m) appears in the description of TSS.

Table 1. Performance comparison

	Security		Typical values	Bitsizes[a]		
	Tightness	Assumption[b]		σ	pk	sk[c]
GHR (basic)	$O(\frac{1}{q_s})$	Div + SRSA	$\ell_n \gg 1024$	ℓ_n	$2\ell_n$	$\frac{1}{2}\ell_n$
GHR (tight)	O(1)	Div + DL + SRSA	$\ell_n = \ell_p = 1024$ $\ell_q = 160$	$\ell_n + \ell_q$	$2\ell_n + 3\ell_p$	$\frac{1}{2}\ell_n$
Twin-GHR	O(1)	SRSA	$\ell_n = 1024$ $\ell_m = 160$	$2\ell_n + 2\ell_m$	$4\ell_n$	ℓ_n
CS	$O(\frac{1}{q_s})$	SRSA	$\ell_n \gg 1024$ $\ell_h = 160$	$2\ell_n + \ell_h$	$3\ell_n + \ell_h$	$\frac{1}{2}\ell_n$
CL	$O(\frac{1}{q_s})$	SRSA	$\ell_n \gg 1024$ $\ell_m = 160$, $\ell = 80$	$2\ell_n + 2\ell_m + \ell$	$4\ell_n$	$\frac{1}{2}\ell_n$
Fischlin	$O(\frac{1}{q_s})$	SRSA	$\ell_n \gg 1024$ $\ell_h = 160$	$\ell_n + 2\ell_h$	$4\ell_n$	$\frac{1}{2}\ell_n$
TSS	O(1)	SRSA	$\ell_n = 1024$ $\ell_m = 160$	$2\ell_n$	$4\ell_n + \ell_m$	ℓ_n

[a] To ease the reading, the bitsizes are rounded up to a few bits.

[b] Div stands for the division intractability assumption and DL for the discrete logarithm assumption.

[c] In the description of GHR and the CS-like schemes (Section 3), we have $\mathsf{sk} = \{p', q'\}$; however, it is possible to only store the value of p' and to recover q' from p' (and pk). Similarly, for Twin-GHR, $\mathsf{sk} = \{p', P'\}$ is sufficient, and for TSS, it is possible to recover $\mathsf{sk} = \{p', q', D\}$ from p', P' (and pk).

implementation. Compared to the former, TSS does not rely on an injective prime generation and needs no prime generation at all in the verification algorithm. Further, TSS offers shorter signatures.

On the minus side, our scheme produces longer signatures than Fischlin or GHR (but shorter than CS or CL). Another drawback is computational. TSS requires the generation of a large random prime. Even using efficient methods (*e.g.*, [JPV00]), this may be time-consuming for low-cost cryptographic devices. We present in the next section an on-line/off-line version of our scheme to address this issue.[5]

4.4 On-Line/Off-Line Version

We present hereafter a variant of our scheme that allows the signer to carry out costly computations *before* knowing the message to be signed. This type of signature scheme is usually referred to as *on-line/off-line scheme* [EGM96, ST01]. Using this paradigm, once the message is known, only a very fast on-line phase is needed. This property is paramount for time-constrained applications or for low-cost smartcards.

[5] We observe that in Twin-GHR, only part of the signature can be precomputed (namely, s_1); parameter s_2 is dependent on the message to be signed.

The message space is $\mathcal{M} = \{0,1\}^{\ell_m}$. Let ℓ_n and ℓ be two security parameters. Typical values are $\ell = 80$ and $\ell_n = 1024$. Our TSS scheme, in its on-line/off-line version, then goes as follows.

Gen: On input ℓ_n and ℓ_m:
- choose an (odd) $(\ell_m + 1)$-bit prime E
- generate two random ℓ_n-bit safe RSA moduli $n = (2p' + 1)(2q' + 1)$ and $N = (2P' + 1)(2Q' + 1)$ such that $\gcd(P'Q', E) = 1$;
- compute $D = E^{-1} \bmod 2P'Q'$;
- choose at random two elements $u \in \mathbb{Z}_n^*$ and $g \in \mathbb{Z}_N^*$.

The public key is $\mathsf{pk} = \{N, n, u, g, E\}$ and the private key is $\mathsf{sk} = \{p', q', D\}$.

Sign (off-line part): To prepare a coupon:
- choose a random prime c in $[(N + 1)/2, N[$;
- pick a random $(\ell_n + \ell_m + \ell)$-bit integer k';
- compute $s = u^{c^{-1} \bmod 2p'q'} \bmod n$ and $r = g^{(k'-D)}\, c^D \bmod N$.

The coupon is (k', r, s).

Sign (on-line part): Let $m \in \{0,1\}^{\ell_m}$ be the message to be signed:
- take a fresh coupon (k', r, s);
- compute $k = k' + D \cdot m$.

The signature on m is $\sigma = (k, r, s) \in [0, 2^{\ell_n+\ell_m+\ell+1}[\times \mathbb{Z}_N^* \times \mathbb{Z}_n^*$.

Verify: Let $\sigma = (k, r, s)$ be a putative signature on message $m \in \{0,1\}^{\ell_m}$. Then:
- compute $c = g^{m+1}\, (r\, g^{-k})^E \bmod N$;
- check that $s^c \equiv u \pmod n$.

If this condition holds then signature σ is accepted.

It is worth remarking that the key generation in the on-line/off-line version is *exactly* the one of the regular version: the public/private keys are the same in both versions.

Security reduction. We now show that this on-line/off-line version *tightly* meets the EUF-CMA security notion under the strong RSA assumption. Actually, we prove that an EUF-CMA adversary $\mathcal{A}^*$ against the on-line/off-line version is an sEUF-CMA adversary against the regular version of our signature scheme. In more detail, given a public key $\widehat{\mathsf{pk}} = \{\hat{N}, \hat{n}, \hat{u}, \hat{g}, \hat{E}\}$ and (at most) q_s chosen-message calls to a TSS signing oracle, we want to produce a TSS signature forgery $\hat{\sigma}_* = (\hat{r}_*, \hat{s}_*)$ on a message $\hat{m}_*$, using $\mathcal{A}^*$.

- We let $\mathsf{pk} = \widehat{\mathsf{pk}}$ (*i.e.*, $\{N, n, u, g, E\} = \{\hat{N}, \hat{n}, \hat{u}, \hat{g}, \hat{E}\}$) as the public key for the on-line/off-line version.
- When $\mathcal{A}^*$ requests an [on-line/off-line] signature on a message $m_j \in \{0,1\}^{\ell_m}$, for $j \in \{1, \ldots, q_s\}$, we call the TSS signing oracle on input message $\hat{m}_j := m_j$ and get back a TSS signature $\hat{\sigma}_j = (\hat{r}_j, \hat{s}_j) \in [0, \hat{N}[\times [0, \hat{n}[$ such that

$$\begin{cases} \hat{c}_j := \hat{g}^{m_j+1}\ \hat{r}_j{}^{\hat{E}} \bmod \hat{N} \text{ is a prime in } [(\hat{N}+1)/2, \hat{N}[, \text{ and} \\ \hat{s}_j{}^{\hat{c}_j} \equiv \hat{u} \pmod{\hat{n}}\ . \end{cases}$$

Next, we pick a random $(\ell_n + \ell_m + \ell)$-bit integer k_j. We compute $r_j = \hat{r}_j\ \hat{g}^{k_j} \bmod \hat{N}$ and let $s_j = \hat{s}_j$. We return $\sigma_j = (k_j, r_j, s_j)$ as the on-line/off-line signature on message m_j.

It is easy to see that σ_j is a valid signature since $c_j := g^{m_j+1}\,(r_j\, g^{-k_j})^E \bmod N = \hat{g}^{m_j+1}\ \hat{r}_j{}^{\hat{E}} \bmod \hat{N} = \hat{c}_j$ is a prime in $[(N+1)/2, N[$ and $s_j{}^{c_j} \equiv \hat{s}_j{}^{\hat{c}_j} \equiv \hat{u} \equiv u \pmod{n}$.

– Eventually, with probability $\varepsilon_{\mathcal{A}^*}$ and within time $\tau_{\mathcal{A}^*}$, $\mathcal{A}^*$ returns an on-line/off-line signature forgery $\sigma_* = (k_*, r_*, s_*)$ on a message $m_* \in \{0,1\}^{\ell_m}$, with $m_* \neq m_j$ for all $j \in \{1, \ldots, q_s\}$.
– From $\sigma_* = (k_*, r_*, s_*)$, we form the signature forgery $\hat{\sigma}_* = (\hat{r}_*, \hat{s}_*)$, where

$$\hat{r}_* = r_*\ \hat{g}^{-k_*} \bmod \hat{N} \quad \text{and} \quad \hat{s}_* = s_*\,,$$

on message $\hat{m}_* := m_*$. Again, it is easy to see that this is a valid signature. Furthermore, as $m_* \neq m_j$, it obviously follows that $(\hat{m}_*, \hat{\sigma}_*) \neq (\hat{m}_j, \hat{\sigma}_j)$, for all $j \in \{1, \ldots, q_s\}$.

Tightness of the reduction. The statistical distance between the k_j's returned by the signature simulation and the k_j's that would be returned by an actual signer is bounded by $2^{-\ell}$, for each signature. Hence, there exists a reduction that succeeds with probability $\varepsilon \geq \varepsilon_{\mathcal{A}^*} - 2^{-\ell} q_s$ and within time $\tau \lesssim \tau_{\mathcal{A}^*} + (q_s + 1)\,\mathrm{O}({\ell_n}^3)$, neglecting the time required to generate random numbers. As the regular version is tightly related to the flexible RSA problem, the on-line/off-line version is tightly EUF-CMA secure under the strong RSA assumption.

EUF-CMA vs. sEUF-CMA. The security proof assumes an EUF-CMA adversary (as opposed to an sEUF-CMA adversary) against our on-line/off-line signature scheme. Even testing the ranges of (k, r, s) in the verification step would not achieve sEUF-CMA security. Indeed, imagine an sEUF-CMA adversary returning a signature forgery $\sigma_* = (k_*, r_*, s_*) \neq \sigma_j$ on message $m_* = m_j$, for some $j \in \{1, \ldots, q_s\}$. Then, the TSS signature forgery $\hat{\sigma}_* = (\hat{r}_*, \hat{s}_*)$ on message $\hat{m}_* = m_*$ returned by the above reduction is not mandatorily a *valid* forgery, *i.e.*, such that $(\hat{m}_*, \hat{\sigma}_*) \neq (\hat{m}_j, \hat{\sigma}_j)$, since $\hat{m}_* = \hat{m}_j$ and $\sigma_* \neq \sigma_j \iff (k_*, \hat{r}_*\ \hat{g}^{k_*} \bmod \hat{N}, \hat{s}_*) \neq (k_j, \hat{r}_j\ \hat{g}^{k_j} \bmod \hat{N}, \hat{s}_j)$ but

$$(k_*, \hat{r}_*\ \hat{g}^{k_*} \bmod \hat{N}, \hat{s}_*) \neq (k_j, \hat{r}_j\ \hat{g}^{k_j} \bmod \hat{N}, \hat{s}_j) \implies (\hat{r}_*, \hat{s}_*) \neq (\hat{r}_j, \hat{s}_j)\ .$$

It is even more apparent with a counter-example: if $\sigma = (k, r, s)$ is a valid on-line/off-line signature on message m so is $\sigma' = (k+1, g\,r \bmod N, s)$ on the *same* message m. Hence, the on-line/off-line version of TSS we describe is not sEUF-CMA secure, but only EUF-CMA secure.

For most cryptographic applications, existential unforgeability is sufficient. Our TSS signature scheme can however be converted into an on-line/off-line

scheme to accommodate *strong* unforgeability (sEUF) by using standard techniques [ST01], at the price of longer — and thus *different* — keys.

5 Conclusion

This paper presented a practical sEUF-CMA signature scheme whose security is solely and tightly related to the SRSA assumption, in the standard model. In contrast to the CS scheme and its variants, the security of our TSS scheme is optimal and, contrary to the GHR scheme, this optimal bound does not result from the use of so-called division-intractable hash functions. Indeed, the TSS scheme does not require the use of hash functions by its very specification. Actually, TSS scheme is much closer, in its properties, to the twinning-based version of GHR, even if constructed in a completely different manner. The main differences between the two schemes lie in the implementation and in the signature size. Moreover, the TSS scheme also comes with an on-line/off-line version for time-constrained applications or low-cost cryptographic devices. Remarkably, this on-line/off-line version uses exactly the same set of keys as the regular version.

References

[BB04] D. Boneh and X. Boyen. Short signatures without random oracles. In *Advances in Cryptology – EUROCRYPT 2004*, LNCS 3027, pp. 56–73. Springer-Verlag, 2004.

[BC92] J. Bos and D. Chaum. Provably unforgeable signatures. In *Advances in Cryptology – CRYPTO '92*, LNCS 740, pp. 1–14. Springer-Verlag, 1993.

[BLS04] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology* **17**(4):297–319, 2004.

[BM92] M. Bellare and S. Micali. How to sign given any trapdoor permutation. *Journal of the ACM* **39**(1):214–233, 1992.

[BNPS03] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology* **16**(3):185–215, 2003.

[BP97] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, LNCS 1233, pp. 480–494. Springer-Verlag, 1997.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pp. 62–73. ACM Press, 1993.

[BR96] ———. The exact security of digital signatures: How to sign with RSA and Rabin. In *Advances in Cryptology – EUROCRYPT '96*, LNCS 1070, pp. 399–416. Springer-Verlag, 1996.

[BSW06] D. Boneh, E. Shen, and B. Waters. Strongly unforgeable signature schemes based on comptutational Diffie-Hellman. In *Public Key Cryptography – PKC 2006*, LNCS 3958, pp. 229–240. Springer, 2006.

[CD96] R. Cramer and I. Damgård. New generation of secure and practical RSA-based signatures. In *Advances in Cryptology – CRYPTO '96*, LNCS 1109, pp. 173–185. Springer-Verlag, 1996.

[CG05] D. Catalano and R. Gennaro. Cramer-Damgård signatures revisited: Efficient flat-tree signatures based on factoring. In *Public Key Cryptography – PKC 2005*, LNCS 3386, pp. 313–327. Springer-Verlag, 2005.

[CGH98] R. Canetti, O. Golreich, and S. Halevi. The random oracle methodology, revisited. In *30th Annual ACM Symposium on Theory of Computing*, pp. 209–217. ACM Press, 1998.

[CL02] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks (SCN 2002)*, LNCS 2676, pp. 268–289. Springer-Verlag, 2002.

[CLP05] J.-S. Coron, D. Lefranc, and G. Poupard. A new baby-step giant-step algorithm and some applications to cryptanalysis. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, LNCS 3659, pp. 47–60. Springer, 2005.

[CN00] J.-S Coron and D. Naccache. Security analysis of the Gennaro-Halevi-Rabin signature scheme. In *Advances in Cryptology – EUROCRYPT 2000*, LNCS 1807, pp. 91–101. Springer-Verlag, 2000.

[Cor00] J.-S Coron. On the exact security of full domain hash. In *Advances in Cryptology – CRYPTO 2000*, LNCS 1880, pp. 229–235. Springer-Verlag, 2000.

[CS00] R. Cramer and V. Shoup. Signature scheme based on the strong RSA assumption. *ACM Transactions on Information and System Security* **3**(3):161–185, 2000.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory* **IT-22**(6):644–654, 1976.

[DN94] C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. In *Advances in Cryptology – CRYPTO '94*, LNCS 839, pp. 234–246. Springer-Verlag, 1994.

[EGM96] S. Even , O. Goldreich, and S. Micali. On-line/off-line digital signatures. *Journal of Cryptology* **9**(1):35–67, 1996.

[Fis03] M. Fischlin. The Cramer-Shoup strong-RSA signature scheme revisited. In *Public Key Cryptography – PKC 2003*, LNCS 2567, pp. 116–129. Springer-Verlag, 2003.

[FO97] E. Fujisaki and T. Okamoto. Statistical zero-knowledge protocols to prove modular polynomial equations. In *Advances in Cryptology – CRYPTO '97*, LNCS 1294, pp. 16–30. Springer-Verlag, 1997.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, LNCS 263, pp. 186–194. Springer-Verlag, 1987.

[GHR99] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology – EUROCRYPT '99*, LNCS 1592, pp. 123–139. Springer-Verlag, 1999.

[GJ03] E.-J. Goh and S. Jarecki. A signature scheme as secure as the Diffie-Hellman problem. In *Advances in Cryptology – EUROCRYPT 2003*, LNCS 2656, pp. 401–415. Springer-Verlag, 2003.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen message attacks. *SIAM Journal of Computing* **17**(2):281–308, 1988.

[Gol86] O. Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In *Advances in Cryptology – CRYPTO '86*, LNCS 263, pp. 104–110. Springer-Verlag, 1986.

[JPV00] M. Joye, P. Paillier, and S. Vaudenay. Efficient generation of prime numbers. In *Cryptographic Hardware and Embedded Systems – CHES 2000*, LNCS 1965, pp. 340–354. Springer-Verlag, 2000.

[KM04] N. Koblitz and A. Menezes. Another look at "provable security". Cryptology ePrint Archive 2004/152, 2004. To appear in Journal of Cryptology.

[KR00] H. Krawczyk and T. Rabin. Chameleon signatures. In *Symposium on Network and Distributed System Security – NDSS 2000*, pp. 143–154. Internet Society, 2000.

[KS06] K. Kurosawa and K. Schmidt-Samoa. New online/offline signature schemes without random oracles. In *Public Key Cryptography – PKC 2006*, LNCS 3958, pp. 330–346. Springer, 2006.

[KW03] J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In *10th ACM Conference on Computer and Communications Security*, pp. 155–164. ACM Press, 2003.

[Mer87] R. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology – CRYPTO '87*, LNCS 293, pp. 369–378. Springer-Verlag, 1987.

[NPS01] D. Naccache, D. Pointcheval, and J. Stern. Twin signatures: An alternative to the hash-and-sign paradigm. In *8th ACM Conference on Computer and Communications Security*, pp. 20–27. ACM Press, 2001.

[NY89] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *21st Annual ACM Symposium on Theory of Computing*, pp. 33–43. ACM Press, 1989.

[PS96] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology – EUROCRYPT '96*, LNCS 1070, pp. 387–398. Springer-Verlag, 1996.

[PV05] P. Paillier and D. Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In *Advances in Cryptology – ASIACRYPT 2005*, LNCS 3788, pp. 1–20. Springer-Verlag, 2005.

[Rom90] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd Annual ACM Symposium on Theory of Computing*, pp. 387–394. ACM Press, 1990.

[RSA78] R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **21**(2):120–126, 1978.

[ST01] A. Shamir and Y. Tauman. Improved online/offline signature schemes. In *Advances in Cryptology – CRYPTO 2001*, LNCS 2139, pp. 355–367. Springer-Verlag, 2001.

[Wat05] B. Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT 2005*, LNCS 3494, pp. 114–127. Springer, 2005.

[Zhu01] H. Zhu. New digital signature scheme attaining immunity against adaptive chosen message attack. *Chinese Journal of Electronics* **10**(4):484–486, 2001.

[Zhu03] ———. A formal proof of Zhu's signature scheme. Cryptology ePrint Archive, Report 2003/155, 2003.

How to Strengthen Any Weakly Unforgeable Signature into a Strongly Unforgeable Signature

Ron Steinfeld[1], Josef Pieprzyk[1], and Huaxiong Wang[1,2]

[1] Centre for Advanced Computing – Algorithms and Cryptography (ACAC)
Dept. of Computing, Macquarie University, North Ryde, Australia
[2] Nanyang Technological University, Singapore
{rons,josef,hwang}@comp.mq.edu.au
http://www.ics.mq.edu.au/acac/

Abstract. Standard signature schemes are usually designed only to achieve *weak* unforgeability – i.e. preventing forgery of signatures on new messages not previously signed. However, most signature schemes are randomised and allow many possible signatures for a single message. In this case, it may be possible to produce a new signature on a previously signed message. Some applications require that this type of forgery also be prevented – this requirement is called *strong* unforgeability.

At PKC2006, Boneh Shen and Waters presented an efficient transform based on any randomised trapdoor hash function which converts a weakly unforgeable signature into a strongly unforgeable signature and applied it to construct a strongly unforgeable signature based on the CDH problem. However, the transform of Boneh et al only applies to a class of so-called *partitioned* signatures. Although many schemes fall in this class, some do not, for example the DSA signature. Hence it is natural to ask whether one can obtain a truly generic efficient transform based on any randomised trapdoor hash function which converts *any* weakly unforgeable signature into a strongly unforgeable one. We answer this question in the positive by presenting a simple modification of the Boneh-Shen-Waters transform. Our modified transform uses two randomised trapdoor hash functions.

Keywords: Digital signature, strong unforgeability, trapdoor hash function, provable security, transform.

1 Introduction

Background. Standard signature schemes are usually designed only to achieve *weak* unforgeability – i.e. preventing forgery of signatures on new messages not previously signed. However, most signature schemes are randomised and allow many possible signatures for a single message. In this case, it may be possible to produce a new signature on a previously signed message. Some applications (such as constructing chosen-ciphertext secure public key encryption schemes [3] and authenticated key exchange protocols [9]) require that this type of forgery also be prevented – this requirement is called *strong* unforgeability.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 357–371, 2007.

At PKC2006, Boneh Shen and Waters [2] presented an efficient transform based on a randomised trapdoor hash function which converts a weakly unforgeable signature into a strongly unforgeable signature, and applied it to construct a strongly unforgeable signature based on the CDH problem. However, the transform of Boneh et al only applies to a class of so-called *partitioned* signatures. Although many schemes fall in this class, some do not, for example the DSA signature [12]. Hence it is natural to ask whether one can obtain a truly generic efficient transform which converts *any* weakly unforgeable signature into a strongly unforgeable one.

Our Result. We answer the above question in the positive by presenting a general efficient transform which uses randomised trapdoor hash functions to strengthen any weakly unforegeable signature into a strongly unforgeable signature. Our transform makes use of *two* randomised trapdoor hash functions (rather than just one in the less general transform of [2]). Like the transform of [2], our transform requires the randomised trapdoor hash functions to be *strongly* collision-resistant (by the word *strong* we mean here that it is even hard to find two randomisers $r \neq r'$ and a message m such that (m, r) and (m, r') are a collision-pair for the randomised hash function, whereas usually only *weak* collision resistance is needed, i.e. it is only hard to find collisions with distinct message inputs). For this purpose, we show that a small modification of the efficient VSH randomised trapdoor function, which was shown to be *weakly* collision-resistant in [5], gives a strongly collision-resistant function which can be used in this application.

Relation to Previous Work. The problem of converting a weakly unforgeable signature into a strongly unforgeable one can be trivially "solved" in two known ways. The first solution is to construct a one-way function from the weakly unforgeable signature scheme, and then apply the generic construction of a strongly unforgeable signature from any one-way function (see Section 6.5.2 in [8]), but this results in a very inefficient scheme. The second trivial "solution" is to completely ignore the original weakly unforgeable scheme, make additional assumptions, and directly construct the strongly unforgeable scheme from those assumptions. For example, strongly unforgeable signature schemes from the Strong RSA assumption [7,6], Strong Diffie-Hellman assumption [1] or Computational Diffie-Hellman in a bilinear group [2] are known. However, these all seem quite strong and non-classical additional assumptions, and do not make use of the given weakly unforgeable signature.

In contrast to the above trivial solutions, our weak-to-strong transform (like the BSW transform [2]) makes non-trivial use of the given weakly unforgeable signature scheme, and efficiently reduces the problem of strengthening it to the problem of constructing a seemingly simpler cryptographic primitive, namely a randomised trapdoor hash function. As evidence for the practicality of our transform, we note that randomised trapdoor hash functions are known to be efficiently constructible under the classical factoring or discrete-log assumptions, whereas no efficient direct constructions for strongly unforgeable signatures based on these classical assumptions are known (without random

oracles). As an example application, we show (in Section 4.2) how to strengthen the standard Digital Signature Algorithm (DSA) [12], assuming only the weak unforgeability of DSA.

2 Preliminaries

Weak and Strong Unforgeability for Signature Schemes. A signature scheme Σ consists of three efficient algorithms: a *key generation* algorithm KG, a signing algorithm S and a verification algorithm V.

The strong and weak unforgeability of a signature scheme Σ are defined using the following game. A key pair $(sk, pk) = \mathsf{KG}(k)$ is generated, and unforgeability attacker A is given the public key pk. A can run for time t and can issue q signing queries $m_1, \ldots, m_q$ to a signing oracle $\mathsf{S}(sk, \cdot)$, which upon each query m_i returns the signature $\sigma_i = \mathsf{S}(sk, m_i)$ to A. At the end, A outputs a forgery message/signature pair (m^*, σ^*). We say that A succeeds in breaking the *strong unforgeability* of Σ if (m^*, σ^*) passes the verification test V with respect to public key pk, and $(m^*, \sigma^*) \neq (m_i, \sigma_i)$ for all $i = 1, \ldots, q$. We say that A succeeds in breaking the *weak unforgeability* of Σ if (m^*, σ^*) passes the verification test V with respect to public key pk, and $m^* \neq m_i$ for all $i = 1, \ldots, q$. A signature scheme Σ is called (t, q, ϵ) *strongly (respectively, weakly) existentially unforgeable under an adaptive chosen-message attack* if any efficient attacker A with run-time t has success probability at most ϵ in breaking the strong (respectively, weak) unforgeability of Σ.

Randomised Trapdoor (Chameleon) Hash Functions [10,13]. A randomised trapdoor hash function scheme consists of three efficient algorithms: a *key generation* algorithm KG_F, a *hash function evaluation* algorithm F, and a *trapdoor collision finder* algorithm TC_F. On input a security parameter k, the (randomised) key generation algorithm $\mathsf{KG}_F(k)$ outputs a secret/public key pair (sk, pk). On input a public key pk, message $m \in \mathcal{M}$ and random $r \in \mathcal{R}$ (here $\mathcal{M}$ and $\mathcal{R}$ are the message and randomness spaces, respectively), the hash function evaluation algorithm outputs a hash value $h = F_{pk}(m; r) \in \mathcal{H}$ (here $\mathcal{H}$ is the hash string space). On input a secret key sk, a message/randomiser pair $(m_1, r_1) \in M \times R$ and a second message $m_2 \in M$, the trapdoor collision finder algorithm outputs a second randomiser $r_2 = \mathsf{TC}_F(sk, (m_1, r_1), m_2) \in R$ such that (m_1, r_1) and (m_2, r_2) constitute a collision for F_{pk}, i.e. $F_{pk}(m_1; r_1) = F_{pk}(m_2; r_2)$.

There are two desirable security properties for a trapdoor hash function scheme $\mathcal{F} = (\mathsf{KG}_F, F, \mathsf{TC}_F)$. The scheme $\mathcal{F}$ is called (t, ϵ) *strongly collision-resistant* if any efficient attacker A with run-time t has success probability at most ϵ in the following game. A key pair $(sk, pk) = \mathsf{KG}_F(k)$ is generated, and A is given the public key pk. A can run for time t and succeeds if it outputs a collision $(m_1, r_1), (m_2, r_2)$ for F_{pk} satisfying $F_{pk}(m_1, r_1) = F_{pk}(m_2, r_2)$ and $(m_1, r_1) \neq (m_2, r_2)$. The scheme $\mathcal{F}$ has the *random trapdoor collision* property if for each fixed secret key sk and fixed message pair (m_1, m_2), if r_1 is chosen uniformly at random from $\mathcal{R}$, then $r_2 \stackrel{\text{def}}{=} \mathsf{TC}_F(sk, (m_1, r_1), m_2)$ has a uniform probability distribution on $\mathcal{R}$.

3 Converting Weak Unforgeability to Strong Unforgeability

We begin by reviewing the Boneh-Shen-Waters (BSW) transform that applies to the class of *partitioned* signatures. We then explain the problem that arises in trying to apply the BSW transform to an arbitrary signature scheme while preserving the security proof, and how we resolve the problem with our generic transform.

3.1 The Boneh-Shen-Waters Transform for Partitioned Signatures

The BSW transform [2] converts any weakly unforgeable *partitioned* signature into a strongly unforgeable signature. First we recall the definition of partitioned signatures from [2].

Definition 1 (Partitioned Signature). *A signature scheme Σ is called* partitioned *if it satisfies the following two properties:*

1. *The signing algorithm S can be split into two deterministic subalgorithms S_1 and S_2, such that a signature $\sigma = (\sigma_1, \sigma_2)$ on a message m using secret key sk can be computed as follows:*
   - *choose a random $\omega \in \Omega_\Sigma$,*
   - *compute $\sigma_1 = \mathsf{S}_1(sk, m; \omega)$ and $\sigma_2 = \mathsf{S}_2(sk; \omega)$ (note that σ_2 does not depend on m),*
   - *return signature $\sigma = (\sigma_1, \sigma_2)$.*
2. *For each m and σ_2, there exists at most one σ_1 such that $\sigma = (\sigma_1, \sigma_2)$ verifies as a valid signature on message m under public key pk.*

The BSW transform converts a partitioned signature scheme $\Sigma = (\mathsf{KG}, \mathsf{S}, \mathsf{V})$ (where the signing algorithm S is partitioned into subalgorithms S_1 and S_2, and the signing algorithm randomness space is denoted Ω_Σ) into a new signature scheme $\Sigma_{\mathsf{BSW}} = (\mathsf{KG}_{\mathsf{BSW}}, \mathsf{S}_{\mathsf{BSW}}, \mathsf{V}_{\mathsf{BSW}})$. The transform also makes use of a randomised trapdoor hash function scheme $\mathcal{F} = (\mathsf{KG}_F, F, \mathsf{TC}_F)$ (where the randomness space is denoted $\mathcal{R}_\mathcal{F}$). We remark that in [2] the authors present their transform with a specific trapdoor hash construction for $\mathcal{F}$ based on the discrete-log problem, but here we present it more generally. The new signature scheme Σ_{BSW} is defined as follows:

1. $\mathsf{KG}_{\mathsf{BSW}}(k)$. On input security parameter k, run $\mathsf{KG}(k)$ to generate a secret/public key pair (sk, pk) for signature scheme Σ, and run $\mathsf{KG}_F(k)$ to generate secret/public key pair (sk_F, pk_F) for trapdoor hash scheme $\mathcal{F}$. The secret and public keys for the new signature scheme Σ_{BSW} are:

$$sk_{\mathsf{BSW}} = (sk, pk_F) \text{ and } pk_{\mathsf{BSW}} = (pk, pk_F).$$

2. $\mathsf{S}_{\mathsf{BSW}}(sk_{\mathsf{BSW}}, m)$. On input message m and secret key $sk_{\mathsf{BSW}} = (sk, pk_F)$, a signature is generated as follows:

(a) choose random $\omega \in \Omega_\Sigma$ and $s \in \mathcal{R}_\mathcal{F}$,
(b) compute $\sigma_2 = \mathsf{S}_2(sk; \omega)$,
(c) compute $\bar{m} = F_{pk_F}(m \| \sigma_2; s)$,
(d) compute $\sigma_1 = \mathsf{S}_1(sk, \bar{m}; \omega)$ and return signature $\sigma = (\sigma_1, \sigma_2, s)$.

3. $\mathsf{V}_{\mathsf{BSW}}(pk, m, \sigma)$. A signature $\sigma = (\sigma_1, \sigma_2, s)$ on a message m is verified as follows:
(a) compute $\bar{m} = F_{pk_F}(m \| \sigma_2; s)$,
(b) return $\mathsf{V}(pk, \bar{m}, (\sigma_1, \sigma_2))$.

The security result proven in [2] can be stated as follows (when generalised to the case of an arbitrary trapdoor hash function in place of the composition of a standard collision-resistant hash function and trapdoor hash function in [2]).

Theorem 1 (Boneh–Shen–Waters). *The signature scheme Σ_{BSW} is (t, q, ϵ) strongly existentially unforgeable, assuming the underlying signature scheme Σ is $(t, q, \epsilon/2)$ weakly existentially unforgeable and the randomised trapdoor hash function $\mathcal{F}$ is $(t, \epsilon/2)$ strongly collision-resistant and has the random trapdoor collision property.*

Intuition. The basic idea of the BSW transform (as also explained in [2]) is that the message-independent signature portion σ_2 of a generated signature is protected from modification by appending it to the message m before hashing with F_{pk_F} and signing with the S_1 algorithm. As a consequence, any 'strong unforgeability' attacks which modify σ_2 will lead to either a collision for the hash function F or a 'weak unforgeability' forgery for the underlying signature scheme. However (to set the scene for our generalised construction) we wish to highlight two important issues and how they were addressed in [2]:

(1) *Security Proof Issues:* Following the above intuition, the security proof involves using the strong unforgeability attacker A on Σ_{BSW} to construct attackers $\mathsf{A}_\mathcal{F}$ and A_Σ against the collision resistance and weak unforgeability of schemes $\mathcal{F}$ and Σ, respectively. But note that to answer A's signing queries:
(1.1) $\mathsf{A}_\mathcal{F}$ needs to be able to simulate signatures of Σ_{BSW} without the trapdoor key sk_F for trapdoor hash scheme $\mathcal{F}$.
(1.2) A_Σ needs to be able to simulate signatures of Σ_{BSW} using the signing algorithm $\mathsf{S}(sk, \cdot)$ as a black box (i.e. without individual access to the internal subalgorithms $\mathsf{S}_1(\cdot, sk)$ and $\mathsf{S}_2(\cdot, sk)$).
(2) *No Protection for σ_1:* Since the σ_1 signature portion is not hashed, it is not protected from modification by the transform.

These issues were addressed in [2] as follows. The issue (1.1) was easily resolved by just using the signing algorithm $\mathsf{S}_{\mathsf{BSW}}$ since the latter does not make use of sk_F. The issue (1.2) was resolved using an alternative signing algorithm which uses the trapdoor key of hash function F_k and the 'sign-then-switch' paradigm [13] to sign using $S(sk, \cdot)$ as a black box (namely, to sign m, one picks a random $s' \in \mathcal{R}_\mathcal{F}$ and an arbitrary string σ_2' and signs $\bar{m} = F_{pk_F}(m \| \sigma_2'; s')$ to obtain $\sigma = (\sigma_1, \sigma_2) = \mathsf{S}(sk, \bar{m})$, and then 'switch' s' to $s = \mathsf{TC}_F(sk_F, (m \| \sigma_2', s'), (m \| \sigma_2))$ using the trapdoor key sk_F, yielding the signature (σ_1, σ_2, s)). Finally, the issue (2) was resolved

vt using Property 2 of partitioned signatures (see Def. 1), which implies that protection of σ_1 is not needed, because one cannot modify σ_1 alone without making the signature invalid.

3.2 Our Generic Transform for Arbitrary Signatures

Intuition. Our goal is to construct a generic transform which can convert any weakly unforgeable signature to a strongly unforgeable one, i.e. we seek a transform which does not rely on the properties of partitioned signatures. Suppose we attempt to modify the BSW transform for this purpose. To address the issue (2) in the previous section, we must protect the *whole* signature from modification. Referring to Fig 1(a), suppose we modify the BSW construction by feeding back the whole signature σ (not just σ_2) into the hash function F_{pk_F} input. The problem is that we obtain a closed loop, where message input $\bar{m}$ of $\mathsf{S}(sk, \cdot)$ depends on the output signature σ. Using the trapdoor key sk_F of the hash function F_{pk_F} and the black box $\mathsf{S}(sk, \cdot)$, we can still produce valid signatures of Σ_{BSW} using the 'sign-then-switch' method outlined in the previous section, but we can no longer produce signatures of Σ_{BSW} without the trapdoor key sk_F (even if we know sk). Therefore, the proof of security for the modified construction collapses due to issue (1.1) discussed in the previous section. Our solution for resolving this issue is to introduce a *second* trapdoor hash function H in this closed loop as shown in Fig. 1(b). This resolves the issue (1.1) by allowing us to use the 'hash-then-switch' method to simulate signatures of Σ_{BSW} using sk_H and sk (without knowing sk_F), or using sk_F and sk (without knowing sk_H), and the last two signing methods produce identically distributed signatures thanks to the 'random trapdoor collision' property of F and H.

Construction. Following the above intuition, we now give a precise description and security proof for our generic transform GBSW. The GBSW transform converts an arbitrary signature scheme $\Sigma = (\mathsf{KG}, \mathsf{S}, \mathsf{V})$ (where the signing algorithm randomness space is denoted Ω_Σ) into a new signature scheme $\Sigma_{\mathsf{GBSW}} = (\mathsf{KG}_{\mathsf{GBSW}}, \mathsf{S}_{\mathsf{GBSW}}, \mathsf{V}_{\mathsf{GBSW}})$. The transform makes use of two randomised trapdoor hash function schemes $\mathcal{F} = (\mathsf{KG}_F, F, \mathsf{TC}_F)$ (with randomness space $\mathcal{R}_\mathcal{F}$) and $\mathcal{H} = (\mathsf{KG}_H, H, \mathsf{TC}_H)$ (with randomness space $\mathcal{R}_\mathcal{H}$). The new signature scheme Σ_{GBSW} is defined as follows:

1. $\mathsf{KG}_{\mathsf{GBSW}}(k)$. On input security parameter k, run $\mathsf{KG}(k)$ to generate a secret/public key pair (sk, pk) for signature scheme Σ, and run $\mathsf{KG}_F(k)$ and $\mathsf{KG}_H(k)$ to generate secret/public key pairs (sk_F, pk_F) and (sk_H, pk_H) for trapdoor hash schemes $\mathcal{F}$ and $\mathcal{H}$, respectively. The secret and public keys for the new signature scheme Σ_{GBSW} are:

$$sk_{\mathsf{GBSW}} = (sk, sk_H, pk_F, pk_H) \text{ and } pk_{\mathsf{GBSW}} = (pk, pk_F, pk_H).$$

2. $\mathsf{S}_{\mathsf{GBSW}}(sk_{\mathsf{GBSW}}, m)$. On input message m and secret key $sk_{\mathsf{GBSW}} = (sk, sk_H, pk_F, pk_H)$, a signature is generated as follows:

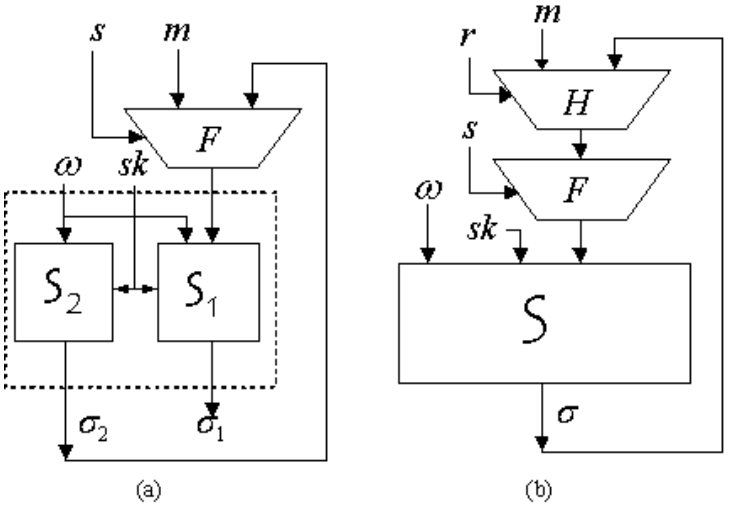

Fig. 1. (a) Boneh–Shen–Waters (BSW) transform for partitioned signature schemes. (b) Our Generalised BSW transform for arbitrary signature schemes.

(a) choose random elements $\omega \in_R \Omega_\Sigma$, $s \in_R \mathcal{R}_\mathcal{F}$, and $r' \in_R \mathcal{R}_\mathcal{H}$,
(b) compute $h = H_{pk_H}(m'\|\sigma'; r')$, for some arbitrary fixed strings m' and σ'.
(c) compute $\bar{m} = F_{pk_F}(h; s)$,
(d) compute $\sigma = \mathsf{S}(sk, \bar{m}; \omega)$,
(e) using trapdoor collision finder for H to compute $r = \mathsf{TC}_H(sk_H, (m'\|\sigma', r'), m\|\sigma)$ such that $H_{pk_H}(m\|\sigma; r) = h$ and return signature $\sigma_{\mathsf{GBSW}} = (\sigma, r, s)$.

3. $\mathsf{V}_{\mathsf{GBSW}}(pk_{\mathsf{GBSW}}, m, \sigma_{\mathsf{GBSW}})$. A signature $\sigma_{\mathsf{GBSW}} = (\sigma, r, s)$ on a message m using public key $pk_{\mathsf{GBSW}} = (pk, pk_F, pk_H)$ is verified as follows:
(a) compute $h = H_{pk_H}(m\|\sigma; r)$,
(b) compute $\bar{m} = F_{pk_F}(h; s)$,
(c) return $\mathsf{V}(pk, \bar{m}, \sigma)$.

Remark 1: We implicitly assume of course, that the verification algorithm $\mathsf{V}_{\mathsf{GBSW}}$ *immediately rejects any signature* (σ, r, s) *for which* $r \notin \mathcal{R}_\mathcal{H}$ *or* $s \notin \mathcal{R}_\mathcal{F}$.

Remark 2: One can set the schemes $\mathcal{F}$ *and* $\mathcal{H}$ *to be identical – the important requirement is that the key pair instances* (sk_F, pk_F) *and* (sk_H, pk_H) *are generated by two independent runs of the key generation algorithms of* $\mathcal{F}$ *and* $\mathcal{H}$*, respectively.*

The following theorem proves the strong unforgeability of the scheme Σ_{GBSW}, assuming the weak unforgeability of the underlying signature scheme Σ.

Theorem 2. *The signature scheme* Σ_{GBSW} *is* (t, q, ϵ) *strongly existentially unforgeable, assuming the underlying signature scheme* Σ *is* $(t, q, \epsilon/3)$ *weakly existentially unforgeable and the randomised trapdoor hash functions* $\mathcal{F}$ *and* $\mathcal{H}$

are both $(t, \epsilon/3)$ *strongly collision-resistant and both have the random trapdoor collision property.*

Proof. Let A denote a (t, q, ϵ) attacker against the strong unforgeability of Σ_{GBSW}. We show how to construct attackers A_Σ, $\mathsf{A}_\mathcal{F}$ and $\mathsf{A}_\mathcal{H}$ against the weak unforgeability of Σ and strong collision-resistance of $\mathcal{F}$ and $\mathcal{H}$, respectively, such that at least one of A_Σ, $\mathsf{A}_\mathcal{F}$ or $\mathsf{A}_\mathcal{H}$ succeeds with probability at least $\epsilon/3$, and all have run-time t, which establishes the theorem.
We first classify the forgery produced by attacker A. Suppose that A is run on input $pk_{\mathsf{GBSW}} = (pk, pk_F, pk_H) = \mathsf{KG}_{\mathsf{GBSW}}(k)$. For $i = 1, \ldots, q$, let m_i denote the ith sign query of A and (σ_i, r_i, s_i) the answer to that query. Furthermore, let $h_i = H_{pk_H}(m_i \| \sigma_i; r_i)$ and $\bar{m}_i = F_{pk_F}(h_i; s_i)$ be the values computed by the signing algorithm in answering the ith sign query. Finally, let $(m^*, (\sigma^*, r^*, s^*))$ denote the output message/signature forgery of A and define $h^* = H_{pk_H}(m^* \| \sigma^*; r^*)$ and $\bar{m}^* = F_{pk_F}(h^*; s^*)$. Let Succ denote the event that A succeeds to break the strong unforgeability of Σ_{GBSW}. If Succ occurs then it easy to see that at least one of the following subevents must occur:

- subevent Succ_I (Type I forgery): $\bar{m}^* \notin \{\bar{m}_1, \ldots, \bar{m}_q\}$,
- subevent Succ_{II} (Type II forgery): there exists $i^* \in \{1, \ldots, q\}$ such that $\bar{m}^* = \bar{m}_{i^*}$ but $(h^*, s^*) \neq (h_{i^*}, s_{i^*})$,
- subevent Succ_{III} (Type III forgery): there exists $i^* \in \{1, \ldots, q\}$ such that $\bar{m}^* = \bar{m}_{i^*}$ and $(h^*, s^*) = (h_{i^*}, s_{i^*})$ but $(m^*, \sigma^*, r^*) \neq (m_{i^*}, \sigma_{i^*}, r_{i^*})$.

Since event Succ occurs with probability ϵ, it follows that one of the above 3 subevents occur with probability at least $\epsilon/3$. Accordingly, our attackers A_Σ, $\mathsf{A}_\mathcal{F}$ and $\mathsf{A}_\mathcal{H}$ described below will each run A and succeed if subevents Succ_I, Succ_{II} and Succ_{III} occur, respectively. In each of those three runs of A we show that the distribution of A's view is perfectly simulated as in the real attack, so that the subevents Succ_I, Succ_{II} and Succ_{III} occur with the same probability as in the real attack, and hence at least one of attackers A_Σ, A_F and A_H succeeds with probability $\epsilon/3$, as claimed.

Attacker A_Σ. The attacker A_Σ against the weak unforgeability of Σ runs as follows on input public key pk (where $(pk, sk) = \mathsf{GK}(k)$ is a challenge key pair for Σ).

Setup. A_Σ runs $\mathsf{KG}_F(k)$ and $\mathsf{KG}_H(k)$ to generate secret/public key pairs (sk_F, pk_F) and (sk_H, pk_H) for trapdoor hash schemes $\mathcal{F}$ and $\mathcal{H}$, respectively, and runs A with input public key $pk_{\mathsf{GBSW}} = (pk, pk_F, pk_H)$.

Sign Queries. When attacker A makes its ith sign query message m_i, A_Σ responds as follows:

- choose random elements $s_i \in_R \mathcal{R}_\mathcal{F}$, and $r'_i \in_R \mathcal{R}_\mathcal{H}$,
- compute $h_i = H_{pk_H}(m' \| \sigma'; r'_i)$, for some arbitrary fixed strings m' and σ',
- compute $\bar{m}_i = F_{pk_F}(h_i; s_i)$,
- query message $\bar{m}_i$ to sign oracle of A_Σ to obtain answer $\sigma_i = \mathsf{S}(sk, \bar{m}_i; \omega)$ for a random $\omega \in_R \Omega_\Sigma$,

– use trapdoor collision finder for H to compute $r_i = \mathsf{TC}_H(sk_H, (m'\|\sigma', r'_i), (m_i\|\sigma_i))$ such that $H_{pk_H}(m_i\|\sigma_i; r_i) = h_i$ and return signature (σ_i, r_i, s_i) to A.

Output. After A outputs its forgery $(m^*, (\sigma^*, r^*, s^*))$, A_Σ computes $h^* = H_{pk_H}(m^*\|\sigma^*; r^*)$ and $\bar{m}^* = F_{pk_F}(h^*; s^*)$, and outputs $(\bar{m}^*, \sigma^*)$ as its forgery for Σ.

Notice that in the above game A_Σ perfectly simulates the real signing oracle $\mathsf{S}_{\mathsf{GBSW}}$ of A (because A_Σ simply follows the real signing procedure, exploiting the fact that $\mathsf{S}_{\mathsf{GBSW}}$ makes only black box access to the signing oracle $\mathsf{S}(sk, \cdot)$ of Σ, and that A_Σ knows the trapdoor key sk_H for H). Furthermore, if A succeeds and outputs a type I forgery, i.e. if subevent Succ_I occurs, then $(\bar{m}^*, \sigma^*)$ verifies as a valid message/signature pair for Σ and $\bar{m}^* \notin \{\bar{m}_1, \ldots, \bar{m}_q\}$, meaning that A breaks the weak unforgeability of Σ, as required.

Attacker $\mathsf{A}_\mathcal{F}$. The attacker $\mathsf{A}_\mathcal{F}$ against the strong collision-resistance of $\mathcal{F}$ runs as follows on input public key pk_F (where $(pk_F, sk_F) = \mathsf{GK}_F(k)$ is a challenge key pair for $\mathcal{F}$).

Setup. $\mathsf{A}_\mathcal{F}$ runs $\mathsf{KG}(k)$ and $\mathsf{KG}_H(k)$ to generate secret/public key pairs (sk, pk) and (sk_H, pk_H) for schemes Σ and $\mathcal{H}$, respectively, and runs A with input public key $pk_{\mathsf{GBSW}} = (pk, pk_F, pk_H)$.

Sign Queries. When attacker A makes its ith sign query message m_i, $\mathsf{A}_\mathcal{F}$ responds with $(\sigma_i, r_i, s_i) = \mathsf{S}_{\mathsf{GBSW}}(sk_{\mathsf{GBSW}}, m_i)$, where $sk_{\mathsf{GBSW}} = (sk, sk_H, pk_F, pk_H)$. $\mathsf{A}_\mathcal{F}$ also stores $(m_i, \sigma_i, r_i, s_i)$ in a table for later use.

Output. After A outputs its forgery $(m^*, (\sigma^*, r^*, s^*))$, $\mathsf{A}_\mathcal{F}$ computes $h^* = H_{pk_H}(m^*\|\sigma^*; r^*)$ and then $\bar{m}^* = F_{pk_F}(h^*; s^*)$, and searches its table of A's queries for entry $i^* \in \{1, \ldots, q\}$ such that $\bar{m}^* = m_{i^*}$ but $(h^*, s^*) \neq (h_{i^*}, s_{i^*})$. If such entry is found, $\mathsf{A}_\mathcal{F}$ outputs strong collision $(h^*; s^*), (h_{i^*}; s_{i^*})$ for $\mathcal{F}$, else $\mathsf{A}_\mathcal{F}$ fails.

In the above game $\mathsf{A}_\mathcal{F}$ perfectly simulates the real signing oracle $\mathsf{S}_{\mathsf{GBSW}}$ of A (because $\mathsf{A}_\mathcal{F}$ knows both sk and sk_H and follows the real signing algorithm). Furthermore, $\mathsf{A}_\mathcal{F}$ succeeds in breaking the strong collision-resistance of $\mathcal{F}$ if A outputs a type II forgery, i.e. if subevent Succ_{II} occurs (because $(h^*, s^*) \neq (h_{i^*}, s_{i^*})$ but $\bar{m}^* = F_{pk_F}(h^*; s^*) = F_{pk_F}(h_{i^*}; s_{i^*}) = \bar{m}_{i^*}$), as required.

Attacker $\mathsf{A}_\mathcal{H}$. The attacker $\mathsf{A}_\mathcal{H}$ against the strong collision-resistance of $\mathcal{H}$ runs as follows on input public key pk_H (where $(pk_H, sk_H) = \mathsf{GK}_H(k)$ is a challenge key pair for $\mathcal{H}$).

Setup. $\mathsf{A}_\mathcal{H}$ runs $\mathsf{KG}(k)$ and $\mathsf{KG}_F(k)$ to generate secret/public key pairs (sk, pk) and (sk_F, pk_F) for schemes Σ and $\mathcal{F}$, respectively, and runs A with input public key $pk_{\mathsf{GBSW}} = (pk, pk_F, pk_H)$.

Sign Queries. When attacker A makes its ith sign query message m_i, $\mathsf{A}_\mathcal{H}$ responds as follows:

– choose random elements $s'_i \in_R \mathcal{R}_\mathcal{F}$, and $r_i \in_R \mathcal{R}_\mathcal{H}$,
– compute $\bar{m}_i = F_{pk_F}(h'_i; s'_i)$, for some arbitrary fixed string h'_i,

– compute $\sigma_i = \mathsf{S}(sk, \bar{m}_i; \omega)$ for a random $\omega \in_R \Omega_\Sigma$,
– compute $h_i = H_{pk_H}(m_i \| \sigma_i; r_i)$,
– use trapdoor collision finder for $\mathcal{F}$ to compute $s_i = \mathsf{TC}_F(sk_F, (h'_i, s'_i), h_i)$ such that $F_{pk_F}(h_i; s_i) = \bar{m}_i$ and return signature (σ_i, r_i, s_i) to A. $\mathsf{A}_\mathcal{H}$ also stores $(m_i, \sigma_i, r_i, s_i)$ in a table for later use.

Output. After A outputs its forgery $(m^*, (\sigma^*, r^*, s^*))$, $\mathsf{A}_\mathcal{H}$ computes $h^* = H_{pk_H}(m^* \| \sigma^*; r^*)$ and searches its table of A's queries for entry $i^* \in \{1, \ldots, q\}$ such that $h^* = h_{i^*}$ but $(m^*, \sigma^*, r^*) \neq (m_{i^*}, \sigma_{i^*}, r_{i^*})$. If such entry is found, $\mathsf{A}_\mathcal{H}$ outputs strong collision $(m^* \| \sigma^*; r^*), (m_{i^*} \| \sigma_{i^*}; r_{i^*})$ for $\mathcal{H}$, else $\mathsf{A}_\mathcal{H}$ fails.

In the above game, $\mathsf{A}_\mathcal{H}$ succeeds in breaking the strong collision-resistance of $\mathcal{H}$ if A outputs a type III forgery, i.e. if subevent Succ_{III} occurs (because $(m^* \| \sigma^*, r^*) \neq (m_{i^*} \| \sigma_{i^*}, r_{i^*})$ but $h^* = H_{pk_H}(m^* \| \sigma^*; r^*) = H_{pk_H}(m_{i^*} \| \sigma_{i^*}; r_{i^*}) = h_{i^*}$), as required.

It remains to show that in the above game, $\mathsf{A}_\mathcal{H}$ perfectly simulates the real signing oracle $\mathsf{S}_{\mathsf{GBSW}}$ of A. For any fixed message m and fixed signature triple $(\widehat{\sigma}, \widehat{r}, \widehat{s})$, let $P_{real}(\widehat{\sigma}, \widehat{r}, \widehat{s})$ denote the probability that the real signing algorithm $\mathsf{S}_{\mathsf{GBSW}}$ outputs $(\widehat{\sigma}, \widehat{r}, \widehat{s})$ for input message m (over the random choices $r' \in_R \mathcal{R}_H$, $s \in_R \mathcal{R}_F$, $\omega \in_R \Omega_\Sigma$ of the real signing oracle). Similarly, let $P_{sim}(\widehat{\sigma}, \widehat{r}, \widehat{s})$ denote the probability that the sign oracle simulator of $\mathsf{A}_\mathcal{H}$ outputs $(\widehat{\sigma}, \widehat{r}, \widehat{s})$ for input message m (over the random choices $r \in_R \mathcal{R}_H$, $s' \in_R \mathcal{R}_F$, $\omega \in_R \Omega_\Sigma$ of the simulator). Then, defining $\widehat{h} = H_{pk_H}(m \| \widehat{\sigma}; \widehat{r})$ and $\widehat{\bar{m}} = F_{pk_F}(\widehat{h}; \widehat{s})$, we have:

$$
\begin{aligned}
&P_{real}(\widehat{\sigma}, \widehat{r}, \widehat{s}) = \\
&\quad \Pr_{r', s, \omega}[(\mathsf{TC}_H(sk_H, (m' \| \sigma', r'), m \| \widehat{\sigma}) = \widehat{r}) \wedge (s = \widehat{s}) \wedge (\mathsf{S}(sk, \widehat{\bar{m}}; \omega) = \widehat{\sigma})] \\
&= \Pr_{r' \in_R \mathcal{R}_H}[\mathsf{TC}_H(sk_H, (m' \| \sigma', r'), m \| \widehat{\sigma}) = \widehat{r}] \cdot \Pr_{s \in_R \mathcal{R}_F}[s = \widehat{s}] \\
&\quad \cdot \Pr_{\omega \in_R \Omega_\Sigma}[\mathsf{S}(sk, \widehat{\bar{m}}; \omega) = \widehat{\sigma}] \\
&= \left(\frac{1}{|\mathcal{R}_\mathcal{H}|}\right) \cdot \left(\frac{1}{|\mathcal{R}_\mathcal{F}|}\right) \cdot \Pr_{\omega \in_R \Omega_\Sigma}[\mathsf{S}(sk, \widehat{\bar{m}}; \omega) = \widehat{\sigma}], \qquad (1)
\end{aligned}
$$

where in the second-last row we used the independence of the r', s, ω and in the last row we used the random trapdoor collision property of H and the uniform distribution of s in $\mathcal{R}_F$ chosen by the real signing algorithm.

On the other hand, for the simulated signatures, we have:

$$
\begin{aligned}
&P_{sim}(\widehat{\sigma}, \widehat{r}, \widehat{s}) = \\
&\quad \Pr_{r, s', \omega}[(r = \widehat{r}) \wedge (\mathsf{TC}_F(sk_F, (h', s'), \widehat{h}) = \widehat{s}) \wedge (\mathsf{S}(sk, \widehat{\bar{m}}; \omega) = \widehat{\sigma})] \\
&= \Pr_{r \in_R \mathcal{R}_H}[r = \widehat{r}] \cdot \Pr_{s' \in_R \mathcal{R}_F}[\mathsf{TC}_F(sk_F, (h', s'), \widehat{h}) = \widehat{s}] \cdot \Pr_{\omega \in_R \Omega_\Sigma}[\mathsf{S}(sk, \widehat{\bar{m}}; \omega) = \widehat{\sigma}] \\
&= \left(\frac{1}{|\mathcal{R}_\mathcal{H}|}\right) \cdot \left(\frac{1}{|\mathcal{R}_\mathcal{F}|}\right) \cdot \Pr_{\omega \in_R \Omega_\Sigma}[\mathsf{S}(sk, \widehat{\bar{m}}; \omega) = \widehat{\sigma}], \qquad (2)
\end{aligned}
$$

where in the second-last row we used the independence of r, s', ω and in the last row we used the random trapdoor collision property of F and the uniform distribution of r in $\mathcal{R}_H$ chosen by the simulator.

Comparing (1) and (2), we conclude that $P_{real}(\widehat{\sigma},\widehat{r},\widehat{s}) = P_{sim}(\widehat{\sigma},\widehat{r},\widehat{s})$, so $\mathsf{A}_{\mathcal{H}}$ perfectly simulates the real signing algorithm, as required.

It follows that at least one of the attackers A_{Σ}, $\mathsf{A}_{\mathcal{F}}$ and $\mathsf{A}_{\mathcal{H}}$ succeeds with probability at least $\epsilon/3$, completing the proof of the theorem. □

Remark (Non-Adaptive to Adaptive Security). It is known [13,11] that randomised trapdoor hash functions can also be used to generically upgrade non-adaptive chosen message attack security to adaptive chosen message attack security for signature schemes. Suppose we start with a weakly unforgeable signature secure against non-adaptive message attack and we wish to upgrade it to a strongly unforgeable signature secure against adaptive message attack. A generic solution is to apply our weak-to-strong transform above followed by the non-adaptive-to-adaptive transform from [13,11]. However, it is easy to see (by modifying the attacker A_{Σ} in our proof of Theorem 2 using the technique in [13,11]) that our GBSW transform simultaneously also achieves non-adaptive-to-adaptive conversion, so there is no need to apply the second transform. Similarly, like the transforms in [13,11], our GBSW transform also gives an 'on-line/off-line' signature scheme, where the only on-line operation is collision-finding for trapdoor hash scheme $\mathcal{H}$ (for this application, $\mathcal{H}$ would have to be chosen appropriately to have a collision-finding algorithm faster than signing algorithm S). Finally, we remark that the 'dual' of the above observation does *not* hold, namely it is easy to see that the non-adaptive-to-adaptive transforms in [13,11] *do not* upgrade weak unforgeability to strong unforgeability in general.

4 Implementation Issues and Application

4.1 Implementation of the Randomised Trapdoor Hash Function

We discuss some possible provably secure concrete implementations of the randomised trapdoor hash functions used in our transform.

Discrete-Log Based Construction. A well known Discrete-log based strongly collision-resistant randomised trapdoor hash function is the Chaum–van Heijst–Pfitzmann (CHP) function [4], also used in [2]. This construction $\mathcal{H}_{DL}$ works in any group G of prime order q where discrete-log is hard. Let g denote a generator for G and let J denote a collision-resistant hash function from $\{0,1\}^*$ to $\mathbb{Z}_q$. The key generation algorithm $\mathsf{KG}_{\mathcal{H}_{DL}}$ chooses $x \in_R \mathbb{Z}_q$ and outputs public/secret key pair $pk_{\mathcal{H}} = (g, g_1 = g^x)$ and $sk_{\mathcal{H}} = x$. Given randomiser $r \in \mathbb{Z}_q$ and message m, we define its hash value $H_{DL}(m;r) = g^r g_1^{J(m)}$. Given a message/randomiser pair (m,r) and a second message m', the collision-finder algorithm computes a second randomiser $r' = r + (J(m) - J(m'))x \bmod q$ such that $H_{DL}(m;r) = H_{DL}(m';r')$. Any 'strong' collision $(m;r) \neq (m';r')$ for H_{DL} (with $r, r' \in \mathbb{Z}_q$) implies that $m \neq m'$ (because g has order q) and hence $x = (r - r')/(J(m') - J(m)) \bmod q$, revealing the discrete-log of g_1 to base g. Hence $\mathcal{H}_{DL}$ is strongly collision-resistant (with randomiser space $\mathbb{Z}_q$) as long as discrete-log is hard in G and J is collision-resistant, and $\mathcal{H}_{DL}$ also has the random trapdoor collision property.

Factoring-based Construction. The above DL-based construction has a fast collision-finding algorithm but relatively slow hash evaluation algorithm. Some constructions based on a standard factorization problem are given in [13]. A variant of the recent VSH randomised trapdoor hash function [5] can also be used and has the opposite performance tradeoff: a fast evaluation algorithm but relatively slow collision-finding algorithm. Although the randomised trapdoor hash function described in [5] is not *strongly* collision-resistant, we show how to easily modify it to achieve this property. The original construction H_{VSH} in [5] has public key $n = pq$, where p,q are primes congruent to 3 modulo 4. The secret key is (p, q). Let $J : \{0,1\}^* \rightarrow \{0,1\}^k$ be a collision-resistant hash function. The randomiser space is $\mathbb{Z}_n^*$. Given message m and randomiser $r \in \mathbb{Z}_n^*$, the hash value is $H_{VSH}(m; r) = (r^2 \prod_{i=1}^{k} p_i^{J(m)_i})^2 \bmod n$, where $J(m)_i$ denotes the ith bit of $J(m) \in \{0,1\}^k$ and p_i denotes the ith prime. Given a message/randomiser pair (m, r) and a second message m', the collision-finder algorithm computes a second randomiser r' such that $H_{VSH}(m; r) = H_{VSH}(m'; r')$ by choosing uniformly at random among the 4 fourth roots of $(r^2 \prod_i p_i^{J(m)_i - J(m')_i})^2 \bmod n$ in $\mathbb{Z}_n^*$. The function H_{VSH} is weakly collision resistant assuming hardness of the factoring-related 'Very Smooth Number Non-Trivial Modular Square-Root' (VSSR) problem, but is not strongly collision-resistant because $(m; (-r \bmod n))$ collides with $(m; r)$ for any m, r. However, the function H'_{VSH} defined in the same way but with randomiser space restricted to $\mathbb{Z}_n^* \bigcap (0, n/2)$ is strongly collision-resistant under the VSSR assumption. This follows from the fact that any quadratic residue in $\mathbb{Z}_n^*$ has two of its square roots less than $n/2$ and two above (the negatives modulo n of each of the first two square-roots). The two square-roots r,r' below $n/2$ are congruent modulo one of the prime factors of n but not modulo the other prime factor, so finding both r and r' is as hard as factoring n (since $\gcd(r' - r, n)$ gives either p or q). The random trapdoor collisions property also is preserved by this modification (note that the modified collision-finder algorithm chooses r' uniformly at random among the two fourth roots of $(r^2 \prod_i p_i^{J(m)_i - J(m')_i})^2 \bmod n$ in $\mathbb{Z}_n^* \bigcap (0, n/2)$).

4.2 Application to Strengthen the Standard Digital Signature Algorithm (DSA)

The Digital Signature Standard [12] (DSA) is an example of a randomised signature scheme which probably does *not* fall within the class of partitioned signature schemes, as noted in [2]. In this scheme, the public key is $(g, y = g^x \bmod p)$, where p is prime and $g \in \mathbb{Z}_p^*$ is an element of prime order q, and $x \in \mathbb{Z}_q$ is the secret key. The signature on message m using randomiser $r \in \mathbb{Z}_q$ is (σ_1, σ_2), where $\sigma_2 = (g^r \bmod p) \bmod q$ and $\sigma_1 = r^{-1}(SHA(m) + x\sigma_2) \bmod q$ (here $SHA : \{0,1\}^* \rightarrow \mathbb{Z}_q$ is the SHA-1 hash function). To verify signature (σ_1, σ_2) on message m under public key (g, y), one checks whether $((g^{SHA(m)} y^{\sigma_2})^{1/\sigma_1} \bmod p) \bmod q$ equals σ_2.

Although DSA clearly satisfies Property (1) of partitioned signatures, it probably does not satisfy Property (2). The reason is that given a valid signature (σ_1, σ_2) on a message m, the number of σ'_1 values such that (σ'_1, σ_2) verifies

as a valid signature on m is the number of elements in the group G of order q generated by g which are congruent to σ_2 mod q. As σ_1' runs through all $q-1$ values of $\mathbb{Z}_q$ except σ_1, we heuristically expect the values of $((g^{SHA(m)}y^{\sigma_2})^{1/\sigma_1} \bmod p) \bmod q$ to behave like $q-1$ independent uniformly random elements in $\mathbb{Z}_q$. This heuristic suggests that with "high probability" of about $1-(1-1/q)^{q-1} \approx 0.63$, we expect there exists at least one other $\sigma_1' \neq \sigma_1$ such that (σ_1', σ_2) is also a valid signature on m. Although we do not know how to efficiently find such 'strong forgeries' for DSA, the fact that DSA is not partitioned means that the BSW transform does not provably guarantee the strong unforgeability of DSA, even assuming that DSA is weakly unforgeable.

Applying our generalised transform to DSA with two CHP [4] randomised trapdoor hash functions for $\mathcal{F}$ and $\mathcal{H}$ based on the hardness of discrete-log in the group G used by DSA, we can construct a strengthened DSA signature which is provably strongly unforgeable, assuming only the weak unforgeability of DSA (which immediately implies the hardness of discrete-log in G and hence the strong collision-resistance of $\mathcal{F}$ and $\mathcal{H}$). The resulting concrete system, called SDSA, is as follows.

1. $\mathsf{KG}_{\mathsf{SDSA}}(k)$. On input security parameter k:
   (a) run DSA key generation on input k to generate a DSA key pair $sk_{\mathsf{DSA}} = (p, q, g, x)$ and $pk_{\mathsf{DSA}} = (p, q, g, y)$, where p is prime, q is a divisor of $p-1$, $g \in \mathbb{Z}_p^*$ is an element of order $q > 2^{159}$, x is uniformly random in $\mathbb{Z}_q$ and $y = g^x \bmod p$,
   (b) choose uniformly random $x_H \in \mathbb{Z}_q$ and compute $v = g^{x_H} \bmod p$,
   (c) choose uniformly random $x_F \in \mathbb{Z}_q$ and compute $u = g^{x_F} \bmod p$,
   (d) the secret and public keys for signature scheme SDSA are:

$$sk_{\mathsf{SDSA}} = (p, q, g, x, v, u, x_H) \text{ and } pk_{\mathsf{SDSA}} = (p, q, g, y, v, u).$$

2. $\mathsf{S}_{\mathsf{SDSA}}(sk_{\mathsf{SDSA}}, m)$. On input message m and secret key $sk_{\mathsf{SDSA}} = (p, q, g, x, x_H)$, a signature is generated as follows:
   (a) compute $h = g^{\eta'} v^{SHA(0)} \bmod p$, for uniformly random $\eta' \in \mathbb{Z}_q$ and fixed bit string 0 (e.g. an all zero byte),
   (b) compute $\bar{m} = g^s u^{SHA(h)} \bmod p$ for uniformly random $s \in \mathbb{Z}_q$.
   (c) compute DSA signature (σ_1, σ_2) on "message" $\bar{m}$, where $\sigma_2 = (g^r \bmod p) \bmod q$ for uniformly random $r \in \mathbb{Z}_q$ and $\sigma_1 = r^{-1}(SHA(\bar{m}) + x \cdot \sigma_2) \bmod q$,
   (d) compute $\eta = \eta' + (SHA(0) - SHA(m \| \sigma_1 \| \sigma_2)) \cdot x_H \bmod q$,
   (e) return signature $\sigma_{\mathsf{SDSA}} = (\sigma_1, \sigma_2, \eta, s)$.
3. $\mathsf{V}_{\mathsf{SDSA}}(pk_{\mathsf{SDSA}}, m, \sigma_{\mathsf{SDSA}})$. A signature $\sigma_{\mathsf{SDSA}} = (\sigma_1, \sigma_2, \eta, s)$ on a message m is verified as follows:
   (a) compute $h = g^{\eta} v^{SHA(m \| \sigma_1 \| \sigma_2)} \bmod p$,
   (b) compute $\bar{m} = g^s u^{SHA(h)} \bmod p$,
   (c) accept only if DSA signature (σ_1, σ_2) verifies on "message" $\bar{m}$, namely accept only if $\sigma_2 = ((g^{SHA(\bar{m})} y^{\sigma_2})^{1/\sigma_1} \bmod p) \bmod q$ holds.

We have:

Corollary 1. *The signature scheme* SDSA *is* (t, q, ϵ) *strongly existentially unforgeable assuming that the* DSA *signature is* $(t, \max(q, 1), \epsilon/6)$ *weakly existentially unforgeable.*

Proof. Applying Theorem 2 to the GBSW transform applied to the DSA signature with two CHP trapdoor hash functions $\mathcal{F}$ and $\mathcal{H}$, we can convert any (t, q, ϵ) attacker against the strong unforgeability of SDSA into a $(t, q, \epsilon/3)$ attacker against the weak unforgeability of DSA or a $(t, \epsilon/3)$ attacker against the strong collision-resistance of $\mathcal{F}$ or $\mathcal{H}$ respectively. In turn, any $(t, \epsilon/3)$ attacker against collision-resistance of $\mathcal{F}$ (or $\mathcal{H}$) can be converted into either a $(t, \epsilon/6)$ attacker against the discrete-log problem in the group generated by g, or a $(t, \epsilon/6)$ attacker against the collision-resistance of SHA. Finally, any $(t, \epsilon/6)$ discrete-log attacker can be easily converted into a $(t, 0, \epsilon/6)$ attacker against weak unforgeability of DSA, while any $(t, \epsilon/6)$ attacker against collision-resistance of SHA can be easily converted into a $(t, 1, \epsilon/6)$ attacker against the weak unforgeability of DSA. So in any case, we can construct a $(t, \max(q, 1), \epsilon/6)$ attacker against weak unforgeability of DSA, which gives the claimed result. □

The SDSA scheme requires an extra computation of two products of two exponentiations each in both verification and signature generation over the DSA scheme, the public key contains two additional elements of $\mathbb{Z}_p$ and the signature contains two additional elements of $\mathbb{Z}_q$. A feature of SDSA which may be of use in practice is that it uses the key generation, signature generation and verification algorithms of DSA as subroutines; hence existing implementations of these subroutines can be used without modification to build SDSA implementations.

5 Conclusion

We presented a modification of the Boneh–Shen–Waters transform to strengthen *arbitrary* weakly unforgeable signatures into strongly unforgeable signatures, and presented applications to the Digital Signature Standard (DSA) with suggested concrete implementations of the randomised trapdoor hash functions needed by our transform.

Finally, we have recently learnt (by private communication with I. Teranishi) that, independently and in parallel with our work, Teranishi, Oyama and Ogata [14] propose a 'weak to strong' unforgeability transform which uses a similar idea to our transform, but is less general in its implementation. In particular, the standard model transform in [14] assumes the hardness of the discrete-log problem, whereas our transform works with any randomised trapdoor hash function (for example, our transform can be used with the efficient factoring-based trapdoor hash function from [5]). On the other hand, the discrete-log based transform in [14] has a more efficient verification algorithm compared to our general transform applied using the discrete-log based trapdoor hash function from [4]. A more efficient transform assuming the random-oracle model along with the discrete-log assumption is also described in [14].

Acknowledgements. The authors would like to thank Duncan Wong for interesting discussions and the anonymous referees for their useful comments. This work was supported by Australian Research Council Discovery Grants DP0663452, DP0451484 and DP0665035.

References

1. D. Boneh and X. Boyen. Short Signatures without Random Oracles. In *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73, Berlin, 2004. Springer-Verlag.
2. D. Boneh, E. Shen, and B. Waters. Strongly Unforgeable Signatures Based on Computational Diffie-Hellman. In *PKC 2006*, volume 3958 of *LNCS*, pages 229–240, Berlin, 2006. Springer-Verlag.
3. R. Canetti, S. Halevi, and J. Katz. Chosen-Ciphertext Security from Identity-Based Encryption. In *Eurocrypt 2004*, volume 3027 of *LNCS*, pages 207–222, Berlin, 2004. Springer-Verlag.
4. D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer. In *CRYPTO '91*, volume 576 of *LNCS*, pages 470–484, Berlin, 1991. Springer-Verlag.
5. S. Contini, A.K. Lenstra, and R. Steinfeld. VSH, an Efficient and Provable Collision-Resistant Hash Function. In *Eurocrypt 2006*, volume 4004 of *LNCS*, pages 165–182, Berlin, 2006. Springer-Verlag.
6. R. Cramer and V. Shoup. Signature Schemes Based on the Strong RSA Assumption. *ACM Transactions on Information and System Security (ACM TISSEC)*, 3:161–185, 2000.
7. R. Gennaro, S. Halevi, and T. Rabin. Secure Hash-and-Sign Signatures Without the Random Oracle. In *EUROCRYPT '99*, volume 1592 of *LNCS*, pages 123–139, Berlin, 1999. Springer-Verlag.
8. O. Goldreich. *Foundations of Cryptography, Volume II.* Cambridge University Press, Cambridge, 2004.
9. J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 110–125, Berlin, 2003. Springer-Verlag.
10. H. Krawczyk and T. Rabin. Chameleon Signatures. In *NDSS 2000*, 2000. Available at `http://www.isoc.org/isoc/conferences/ndss/2000/proceedings/`.
11. K. Kurosawa and S. Heng. The Power of Identification Schemes. In *PKC 2006*, volume 3958 of *LNCS*, pages 364–377, Berlin, 2006. Springer-Verlag.
12. National Institute of Standards and Technology (NIST). *Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2*, January 2000.
13. A. Shamir and Y. Tauman. Improved Online/Offline Signature Schemes. In *CRYPTO 2001*, volume 2139 of *LNCS*, pages 355–367, Berlin, 2001. Springer-Verlag.
14. I. Teranishi, T. Oyama, and W. Ogata. General Conversion for Obtaining Strongly Existentially Unforgeable Signatures. In *INDOCRYPT 2006*. To Appear.

Public Key Cryptography and RFID Tags

M. McLoone[1] and M.J.B. Robshaw[2]

[1] Institute of Electronics, Communications, and Information Technology
Queen's University, Belfast, U.K.
M.McLoone@ecit.qub.ac.uk
[2] France Telecom Research and Development
38–40, rue du Général Leclerc
92794 Issy les Moulineaux, Cedex 9, France
matt.robshaw@francetelecom.com

Abstract. When exploring solutions to some of the formidable security problems facing RFID deployment, researchers are often willing to countenance the use of a strong symmetric primitive such as the AES. At the same time it is often claimed that public key cryptography cannot be deployed on low-cost tags. In this paper we give a detailed analysis of the GPS identification scheme. We show that with regards to all three attributes of space, power, and computation time, the on-tag demands of GPS identification compare favourably to the landmark AES implementation by Feldhofer *et al.*. Thus, assumed limits to implementing asymmetric cryptography on low-end devices may need to be re-evaluated.

1 Introduction

The problem of deploying cryptographic mechanisms when memory, power, or processing capability is limited is as old as cryptography itself. For most contemporary applications in resource-rich environments, the performance of the cryptographic primitive is typically of little concern. However, recent interest in *radio frequency identification (RFID) tags* has once again brought the issue of performance to the fore.

The radio-frequency-identification provided by an RFID tag is not new. But recent advances now allow small devices that provide this functionality to be manufactured cheaply. This opens the door to widespread deployment. However the potential for a pervasive deployment of such devices raises many security and privacy concerns and some applications would benefit from, or indeed require, cryptographic functionality.

In reality, the term "RFID tags" covers a range of devices with different computational capabilities. However the term "tag" tends to imply that we are working with some of the less capable devices. At the most limited end of the range, *e.g.* typical EPC Type 1 tags [4], there is in fact little that one can do. Even the most exotic cryptographic proposals from recent years are unsuitable. Instead, perhaps unwittingly, researchers are assuming the use of slightly more sophisticated devices. As we will see in Section 4, one important restriction when

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 372–384, 2007.

using algorithms in constrained environments is the amount of physical space that is required. To aid comparison across different manufacturing technologies, this is usually measured in terms of *gate equivalents* (GE). In practice, equally important considerations are the peak and average power consumption. If these are high then the practical effectiveness of a passive tag can be severely degraded since its power is derived from the reader.

While one would like to use strong, established cryptography in principle, the physical demands of cheaper tags prevent this. One significant paper in recent years [6] has detailed an optimised implementation of the AES [17]. This demonstrated that strong, established cryptography, is within reach of (reasonably) cheap RFID deployment. By contrast, the possibility of implementing an asymmetric scheme on an RFID tag is often dismissed out of hand [1,14]. While such a view is reasonable for most asymmetric schemes, it does not apply to all. As we will show, there are variants of the GPS scheme that require fewer on-tag resources than the AES. Thus, if one is willing to consider the deployment of the AES on a tag, one should also be willing to consider the deployment of asymmetric cryptography under similarly demanding conditions.

In its basic form, the GPS identification scheme [7,19] already has the potential for low on-tag demands. However, a range of optimisations in the literature have extended this potential. This paper, therefore, provides the following:

1. We provide an overview of the GPS identification scheme and its optimisations. Publications on GPS span several years and so we use this paper to bring together much of the work on this topic.
2. We provide the results of a detailed hardware performance analysis of the GPS identification scheme. This is essential in understanding the suitability of any cryptographic proposal for RFID deployment.
3. We outline a range of deployment costs and requirements for an implementation of the GPS scheme in an RFID tag application.

Overall, this paper is concerned with all the practical aspects of implementing and deploying a public key solution on RFID tags.

2 The GPS Identification Scheme

GPS is a public key identification scheme due to Girault, Poupard, and Stern [7,19]. It is standardised and features within ISO/IEC 9798-5 [12] and also appears in the final NESSIE portfolio [13]. One feature of the GPS scheme is its flexibility and there are many variants and optimisations. We will illustrate the essential elements of the GPS scheme using an RSA-like modulus, as featured in most descriptions of GPS and in ISO and NESSIE documentation [12,13], but also in an elliptic curve variant [8] that is likely to be of greater interest in practice.

The standardised form of the GPS scheme is shown in Figure 1. A cryptographic hash function HASH gives outputs of length h, and we use parameters ρ, δ, and σ to denote three particular bit lengths. These values will depend on the intended security level of the scheme, and following the specifications of GPS it

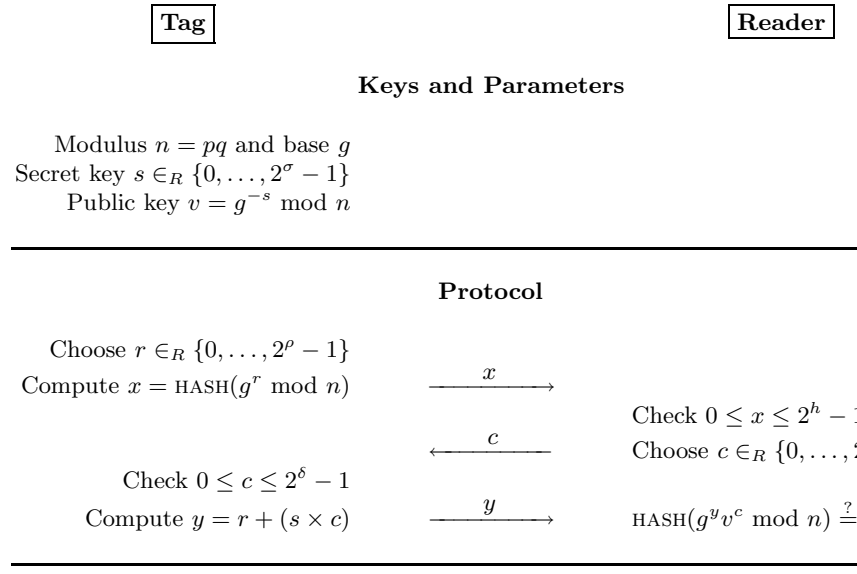

Fig. 1. Overview of the ISO-standardised variant of the GPS identification protocol where h denotes the length of the output from HASH. Some minor protocol details, including opportunities for shared system parameters, are omitted and specific choices for σ, ρ, δ, and the size of n depend on the desired security level.

is typical to set $|r| = \rho = \delta + \sigma + 80$. We provide estimates for a range of parameter values but, by way of illustration, a basic set of parameter values might be $|c| = \delta = 20$, $|s| = \sigma = 160$, and $|r| = \rho = \delta + \sigma + 80 = 260$. Naturally, these parameters can be changed for a range of security/performance trade-offs.

Our focus will be on the workload for the tag (the prover) and we will assume that the reader (the verifier) is more powerful. Initially the GPS scheme in Figure 1 might not appear to be a promising candidate for compact implementation. The reader might be concerned about the large numbers that appear when we use RSA-like moduli. This is avoided by moving to an elliptic-curve based version of GPS, see Figure 2. Further, the first on-tag computation consists of an expensive operation (either exponentiation in Figure 1 or elliptic curve addition in Figure 2) while the second operation is a hash function computation. However, we can remove both operations by using *coupons* [8]. At initialisation, we pre-compute and store t coupons that consist of the pairs (r_i, x_i) for $1 \leq i \leq t$, as shown in Figure 2. While there is a storage cost in using coupons (see Section 5) the performance benefits are immediate. The on-tag performance will be dominated by the simple integer (non-modular) calculation $y = r + (s \times c)$ which is identical in both the classical and the elliptic curve variants of GPS. Here we consider the cost of this operation as well as a range of computation and storage optimisations proposed in the literature.

Tag		Reader
Keys and Parameters		
Curve $\mathcal{C}$ and base point P		Curve $\mathcal{C}$ and base point P
Secret key $s \in_R \{0, \ldots, 2^\sigma - 1\}$		
Public key $V = -sP$		Public key $V = -sP$
Precomputation		
For $0 \leq i \leq t-1$		
Choose $r_i \in_R \{0, \ldots, 2^\rho - 1\}$		
Compute $x_i = \text{HASH}(r_i P)$		
Store coupon (r_i, x_i)		
Protocol		
Select coupon (r_i, x_i)	$\xrightarrow{x_i}$	
		Check $0 \leq x_i \leq 2^h - 1$
	$\xleftarrow{c}$	Choose $c \in_R \{0, \ldots, 2^\delta - 1\}$
Check $0 \leq c \leq 2^\delta - 1$		
Compute $y = r_i + (s \times c)$	$\xrightarrow{y}$	$\text{HASH}(yP + cV) \stackrel{?}{=} x_i$

Fig. 2. Overview of the elliptic curve-based variant of the GPS identification protocol. We use precomputation and coupons where h denotes the length of the output from HASH. Some minor protocol details are omitted and specific choices for different parameters depend on the desired security level.

3 GPS Security and Implementation

The security of the GPS scheme is explored in a range of papers but particularly in [19]. The GPS scheme is a zero-knowledge identification scheme and can be converted into a signature scheme using established methods [15]. However it is the identification scheme that is of interest to us here.

To give some idea of the parameter sizes we might need we consider the options for an adversary. An attacker might attempt to recover the tag-specific secret key s from the public information, but brute-force efforts require a square-root work effort [9]. Thus, for a security level of 80 bits we set $|s| = \sigma = 160$. Given the nature of some RFID applications, other security levels may be acceptable and would allow a corresponding change to the size of s.

Independently of attempts to compromise the per-tag private key, an attacker may attempt to impersonate the tag. The probability of success for an impersonating attacker is determined by the ease with which the challenge

c can be anticipated. Consequently we need to be sure that the set of possible challenges is sufficiently large to deter guessing. Again, much depends on the application. A per-challenge security level of 2^{-8} is conceivable, but 2^{-20} and 2^{-32} would perhaps be more typical. Taking all these issues into consideration, the different parameters we consider in this paper are $\sigma \in \{128, 160\}$, $\delta \in \{8, 20, 32\}$, and for the six possible sets that result we set $\rho = \sigma + \delta + 80$. The importance of generating r in a sound manner is highlighted in [13].

Considering the role and placement of different components, the most appropriate way for the reader to access public keys and shared parameters will be architecture-dependent, see Section 6. However, when using coupons, specific choices in this regard will have no impact on tag performance. Instead, the quantities that will have the most bearing on tag performance are the following:

s	Tag-specific secret key	Stored on tag
r_i, x_i	Per-use commitment	Stored/generated on tag
c	Challenge	Transmitted to tag
y	Response	Computed on tag and transmitted to reader

4 Architectures for Constrained Environments

Two significant restrictions on the suitability of cryptographic algorithms for constrained environments are limitations to space and power. Estimates vary on exactly what space resources might be available, but a consensus is summarised in [14,20] where it is suggested that low-end tags might have a total gate count of around 1000 to 10000 GE of which only around 200-2000 would be available for security features. Passive RFID tags are powered by interaction with the electromagnetic field generated by the reader. Thus, implementations with high average- or peak-power consumption will degrade the practical use of the tag. One way to reduce power consumption is to reduce the clocking rate on the tag. However long computations might fall foul of higher-level protocols.

A landmark paper from CHES 2004 detailed a highly optimised implementation of the AES [6]. This is intended for RFID deployment and makes a useful reference point against which to benchmark other implementations. Therefore we will provide the results of different implementations of the GPS identification scheme and compare these to the highly optimised AES implementation in [6].

4.1 The AES Target Architecture

The AES can be used in a challenge-response authentication protocol and, as a result, an on-tag implementation only requires encryption [6]. While the AES implementation requires[1] 3595 GE and 8.15 μA, this requires clocking the tag at 100 kHz. This impacts higher-level communication protocols since the prompt

[1] Target requirements were for an implementation of less than 5000 GE with a maximum current consumption of 15 μA.

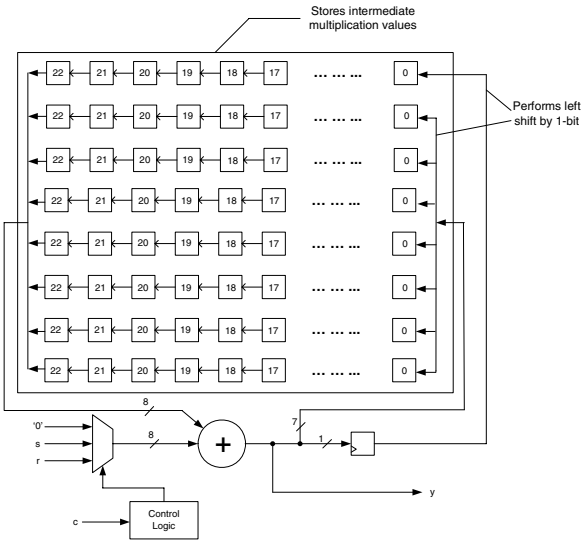

Fig. 3. An 8-bit architecture for the GPS identification scheme implemented with parameters $|s| = \sigma = 160$, $|c| = \delta = 20$, and $\rho = 260$. Some performance characteristics are given in Table 1.

from a reader should be answered within 320 μs and at 100 kHz this implies that a tag response is required in $320 \times 10^{-6} \times 10^5 = 32$ clock cycles. However 32 clock cycles is not enough to encrypt a challenge using the AES, so a modified communication protocol was proposed which allowed the interleaved authentication of multiple tags and provided an individual tag up to 1800 clock cycles to perform the computation [6]. The implementation of Feldhofer *et al.* separates out considerations such as the analogue interface, digital control, and the use of EEPROM for the storage of tag-specific ID and key information [6]. We will follow this example and restrict our attention to the cryptographic computation itself. While coupons have an obvious impact on storage, this is addressed in Section 5.

4.2 GPS Architectures

In our proposed GPS architectures, we assume the use of coupons (see Sections 2 and 5.2) and so the main on-tag computation is $y = r + (s \times c)$. A number of low area and low power architectures have been developed for a range of security levels. We provide implementations for all six parameter sets outlined in Section 3, namely $\sigma \in \{128, 160\}$, $\delta \in \{8, 20, 32\}$, and $\rho = \sigma + \delta + 80$.

An 8-bit architecture is illustrated in Figure 3. This performs byte-wise operations on the inputs s and r and bit-wise operations on the challenge c. To illustrate, consider $\sigma = 160$ and $\delta = 20$ for which we set $\rho = 260$. The overall multiplicand of $(s \times c)$ is 180 bits in length and multiplication can be implemented using the basic left-shift SHIFT-AND-ADD multiplication algorithm [18]. When the input challenge bit is `1` the multiplicand is shifted left by one bit

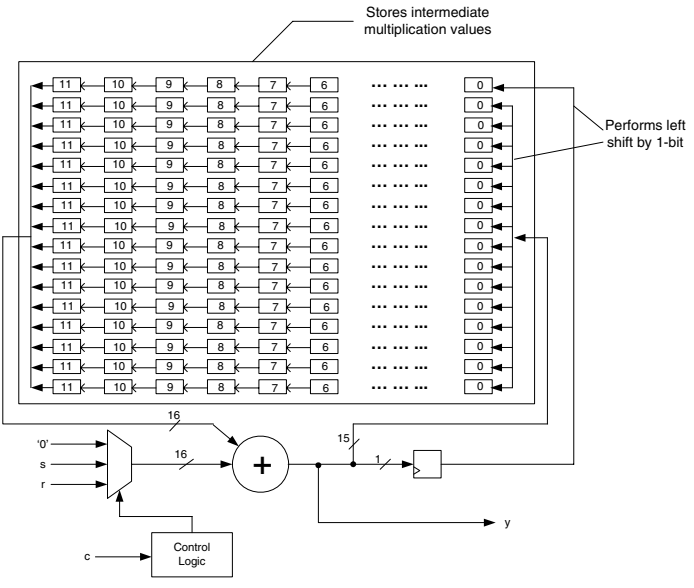

Fig. 4. A 16-bit architecture for the GPS identification scheme implemented with parameters $|s| = \sigma = 160$, $|c| = \delta = 20$, and $\rho = 260$. Some performance characteristics are given in Table 1.

position and the result added to the value of s. When the challenge bit is 0 the multiplicand is shifted left but no addition is performed. A low-area and low-power architecture of this SHIFT-AND-ADD algorithm is designed using an array of shift registers to store the intermediate multiplicand values. These shift registers are also used to perform the shifting required in the algorithm as outlined in Figure 3. In the 8-bit architecture, values of $\delta \leq 24$ can be accommodated and as such, the architecture comprises an array of $23 \times$ 8-bit shift registers. An 8-bit adder is used initially to perform the addition step of the left-shift multiplication algorithm and once the full 180-bit multiplicand is achieved, to reduce area costs, the adder is re-used to perform the byte-wise addition with r.

Figure 4 shows a 16-bit architecture in which s and r are considered as 16-bit blocks. The 16-bit architecture operates in a similar manner to the 8-bit architecture except that a 16-bit adder is used to perform addition within the multiplication and during addition with r. For both architectures, the size of the shift register array will vary according to the input parameters as shown below.

$\delta \in \{8, 20, 32\}$	8	20	32	8	20	32
$\sigma = 128$	17	19	20	9	10	10
$\sigma = 160$	21	23	24	11	12	12
	# 8-bit registers			# 16-bit registers		

Table 1. Some initial implementation results for the GPS identification scheme and the architectures illustrated in Figures 3 and 4

		Area (gates)	Current (μA@100 kHz)	Timing (cycles)
AES [6]		3595	8.15	1016
GPS 8-bit architecture				
$\|s\| = \sigma$	$\|c\| = \delta$			
160	32	1541	3.07	802
128	32	1327	2.62	670
160	20	1486	2.90	585
128	20	1286	2.43	485
160	8	1371	2.72	199
128	8	1167	2.28	163
GPS 16-bit architecture				
$\|s\| = \sigma$	$\|c\| = \delta$			
160	32	1642	3.41	401
128	32	1411	2.93	335
160	20	1642	2.98	401
128	20	1411	2.56	335
160	8	1511	2.53	192
128	8	1298	2.22	158

4.3 Performance Results

The on-tag GPS operation was implemented using the 8-bit and 16-bit architectures of Figures 3 and 4 with UMC 180 nm CMOS technology. The designs were tested using Modelsim and synthesised using Synopsys Physical Compiler. The current estimates for each design were obtained from Synopsys PrimePower and calculated as the average plus twice the standard deviation for a set of randomly generated input values. Table 1 outlines the area, current, and timing measurements for both architectures and all six parameter sets. The required area for all variants lies within the assumed resource constraint of 2000 gates for low-end RFID tags, however there are some additional considerations (see Section 6). With regards to the current consumption, even the largest design in Table 1 requires only 3.41 μA which easily satisfies the 15 μA limit in [6].

If we were to use the interleaved challenge response protocol proposed by Feldhofer *et al.* then a tag must respond 18 ms after a challenge is issued by a reader [6]. Under the assumption that a protocol with similar demands is used for an implementation of the GPS identification scheme, the tag must perform its computation in 1800 clock cycles at a clock frequency of 100 kHz. This is achieved by all the architectures described in Table 1 with room to spare. Interestingly, the power consumption for some variants is so modest that one might consider a faster clocking rate on the tag. Then the response time for some GPS variants could approach the 320 μs required by the basic communication protocol.

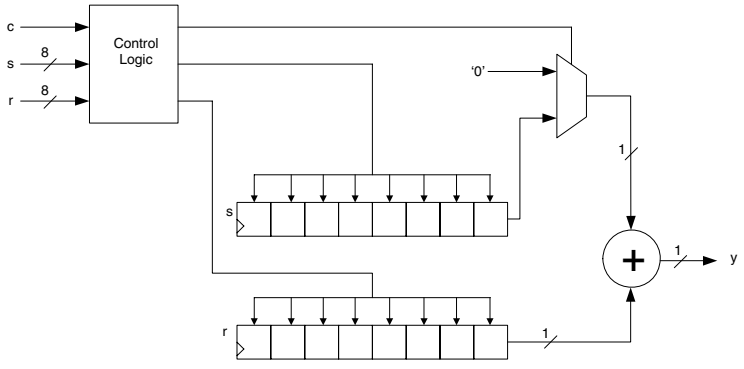

Fig. 5. A basic architecture for the LHW challenge optimisation to GPS [9]. Some performance characteristics are given in Table 2.

5 GPS Optimisations

The literature offers several optimisations to the basic GPS scheme. Broadly speaking they aim to optimise the on-tag computation time or the storage requirements when using coupons.

5.1 Changing the Computation Profile

Girault and Lefranc [9] proposed two ways to reduce the on-tag computational cost. The first required a change to the form of the secret key s, and the second a change to the form of the challenge c. Both approaches turn multiplication into simple serial additions.

Changing the form of the secret key requires some care. The proposal in [9] was subsequently cryptanalysed by Coron, Lefranc, and Poupard [3]. While a patch was proposed, the remaining secure alternatives do not appear to yield attractive performance benefits. We do not consider this proposal further here.

The second approach, however, remains very interesting. This requires using a long challenge c with a low Hamming weight, such that the non-zero bits are at least $\sigma - 1$ zero bits apart (recall that σ is the length of s the secret key). We refer to this as a *low Hamming weight (LHW) challenge*. Such a challenge reduces the on-tag computation to a serial addition of r with a modest number of repetitions of s. The parameters to achieve a security level consistent with a probability of impersonation of 2^{-32} are proposed in [9] and constitute a challenge of length $\delta \approx 850$ with Hamming weight five.

The price to pay when reducing the tag computation in this way is an increase to δ and ρ, namely the size of the challenge and the pre-commitment. While this appears to induce a communication and storage overhead, this is not the case in practice. Even though the challenge is around 850 bits in length, there are compact representations, of around 50 bits, that for instance merely list the positions of the five ones. Moreover, if we use coupons together with a suitably lightweight

Table 2. Some initial performance results using the LHW optimisation [9] of the GPS identification scheme. The parameter values match the security levels provided by $\sigma = 160$ and $\delta = 32$ in the basic scheme.

	Area (gates)	Current (μA@100 kHz)	Timing (clock cycles)
AES [6]	3595	8.15	1016
GPS 8-bit architecture	1541	3.07	802
GPS 16-bit architecture	1642	3.41	401
GPS LHW optimisation	317	0.61	1088

pseudo-random number generator (PRNG) to regenerate the r_i, as proposed in ISO 9798-5 [12], then there would no impact on the required storage [9], see Section 5.2.

We now consider the performance impact of using a LHW challenge c for the computation $y = r + (s \times c)$. A basic architecture for GPS with the LHW challenge is shown in Figure 5. The form of the challenge is described above and for this implementation we use the set of parameters, $\sigma = 160$, $\delta = 848$, and $\rho = 1088$ as proposed in [9]. The values s and r are input in 8-bit blocks, stored in shift registers and operated on serially using a 1-bit adder. The register containing the r byte is reloaded every eight clock cycles with the next byte of r. The register containing the s byte is reloaded with the next s byte when $c = \texttt{1}$ and every subsequent eight cycles. The input bit of c determines when s is to be added to r; addition is performed when $c = \texttt{1}$.

Table 2 provides area, current, and timing measurements for the LHW architecture and compares these with the previous designs. The advantage of the LHW optimisation [9] is clear and, as can be seen in Table 2, the LHW architecture has very low area and current requirements. However, since s and r are added serially the overall computation time for a basic architecture takes 1088 clock cycles. As it stands, this essentially matches the AES implementation, but there are two ways to address the time for the computation. First, given the low current consumption, one could consider a higher clocking rate. While the cycle count would remain unchanged, the elapsed time for the computation would be reduced. Second, it appears to be possible to reduce the number of cycles required by using slightly more sophisticated designs. These require some additional management of the challenge and secret, but early estimates suggest the computation time could be reduced significantly at the expense of increased area. This is the topic of ongoing research.

5.2 Reducing the Storage Requirements

While EEPROM requirements are less of an immediate concern, storing coupons costs space and therefore raise the cost of an deployment. Considering the scheme presented in Figure 2, we would expect each coupon (r_i, x_i) to be $\rho + h$ bits in length, where h is the length of the output from HASH. Since the size of the coupons directly influences the memory requirements of the tag or, conversely,

Table 3. Coupon storage estimates (in bits) for different GPS optimisations. With the LHW optimisation [9], a PRNG as proposed in ISO 9798-5 [12] becomes a necessity.

	Without PRNG			*With PRNG [12]*		
Number of coupons	5	10	20	5	10	20
Basic implementation	2160	4320	8640	800	1600	3200
Optimisation [10]	1710	3420	6840	350	700	1400
Optimisation [8]	322	644	1288	250	500	1000
LHW [9] + *Optimisation* [8]	5690	11380	22760	250	500	1000

the amount of memory limits the number of coupons that can be stored, several optimisations address this issue. Girault and Stern [10] consider the role of the hash function in a range of identification protocols and propose smaller values for the hash output. By considering the capabilities of the adversary, further improvements to the sizes of x_i are proposed by Girault in [8].

Independently, another optimisation reduces the coupon storage requirements by regenerating the r_i instead of storing each value. In ISO documentation [12] it is suggested that the r_i could be generated by a PRNG and so, depending on the size of ρ and the number of coupons required, it might be more effective to remove the contribution of r_i from the coupons at a fixed cost of implementing the PRNG. Within the eSTREAM project [5] some promising stream cipher proposals aimed at efficient hardware implementation appear to occupy around 1500 GE and have very low current consumption [11]. Meanwhile estimates in [2] suggest that dedicated options for a PRNG might require as little as 1000 GE. Thus the burden of implementing a PRNG is unlikely to be overwhelming, and when using the LHW challenge variant of GPS (see Table 2) the total space required for a complete implementation of GPS and a PRNG is likely to be no more than 2000 GE.

Throughout it is important to bear in mind the essential qualities of typical RFID tag deployments. Tags are intended to be cheap and disposable, and there are many applications, *e.g.* event ticketing or visa stamps, where only a handful of coupons might be required over the (short) lifetime of the tag. Thus a PRNG may not be required for many applications.

5.3 Changing the Security Level

As we saw in Section 4.2, basic implementations of the GPS identification scheme are more efficient on-tag than the AES with regards to all three measures of space, power, and computation. As is usually the case, different performance characteristics can be obtained by varying some of the parameter sizes.

Throughout we have aimed to provide estimates for GPS at the 80-bit security level. In this way we match typical basic security recommendations as well as the public-key benchmark of 1024-bit RSA. It is, however, very difficult to assess an appropriate level of security for an RFID application. RFID tags are not particularly secure and they would not be deployed in applications with high security requirements. While the AES offers 128-bit security, this is likely to be

far larger than required for RFID deployment in the foreseeable future. When inviting candidate stream ciphers suitable for low-resource hardware implementation, the eSTREAM project [5] required that they offer 80-bit security. Thus, for certain applications, there is a feeling that 80-bit security may well be adequate. Nevertheless, increasing GPS parameters to give, say, a 128-bit security level leaves the essential message of the paper unchanged. Of course, there may also be applications for which a lower security level is appropriate.

6 Other Issues

Neither a symmetric nor an asymmetric cryptographic deployment is necessarily better than the other. Instead, both have advantages and disadvantages and their relative suitability will depend on the application. That said, there are situations where asymmetric cryptography on an RFID tag can open up new application areas; particular so when we consider the more open deployments we might anticipate for RFID tags. Like any cryptographic algorithm GPS requires a supporting infrastructure, and while it is essentially an architectural issue to decide how best to present the public parameters and key information to the reader, some proposals have been made [8].

The GPS identification scheme is a commitment-challenge-response scheme which may carry some communication complexity. However there is much experience in implementing such protocols and some recent proposals intended for RFID deployment, such as HB+ [14], adopt a similar structure. The problem of an adversary maliciously consuming coupons as a denial of service attack is out of the scope of this paper, but is considered in [2]. Finally, while the resistance of an RFID tag to different forms of side-channel cryptanalysis might not be great, when considered in the framework of a cost-benefit analysis such attacks are rarely worthwhile for the vast majority of RFID tag applications.

7 Conclusions

It is often said that strong public key cryptography cannot be deployed on cheaper RFID tags. In this paper we have investigated the implementation of the GPS identification scheme. With regards to all three attributes of space, power, and computation time, GPS can be implemented as efficiently as today's most efficient AES implementation. Thus, any assumed limits to implementing asymmetric cryptography on low-end RFID tags may need to be reassessed.

Acknowledgements

We thank Marc Girault and Loïc Juniot for many interesting discussions.

References

1. G. Avoine. Cryptography in Radio Frequency Identification and Fair Exchange Protocols. Ph.D. thesis. December 2005.
   Available via `http://lasecwww.epfl.ch/~gavoine/rfid/`.
2. B. Calmels, S. Canard, M. Girault, and H. Sibert. Low-cost Cryptography for Privacy in RFID Systems. In J. Domingo-Ferrer, J. Posegga, and D. Schreckling, editors, *Smart Card Research and Applications, Proceedings of CARDIS 2006*. LNCS 3928, pages 237–251, Springer Verlag, 2006.
3. J.S. Coron, D. Lefranc, and G. Poupard. A New Baby-Step Giant-Step Algorithm and Some Applications to Cryptanalysis. In J. Rao and B. Sunar, editors, Proceedings of CHES 2005, LNCS 3659, pages 47–60, Springer Verlag, 2005.
4. EPCglobal Inc. Home Page. Available via `http://www.epcglobalinc.org/`.
5. eSTREAM Project. Available via `http://www.ecrypt.eu.org/estream/`.
6. M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong Authentication for RFID Systems Using the AES Algorithm. In M. Joye and J.-J. Quisquater, editors, Proceedings of CHES 2004, LNCS 3156, pages 357–370, Springer Verlag, 2004.
7. M. Girault. Self-certified Public Keys. In D. Davies, editor, *Proceedings of Eurocrypt '91*, LNCS 547, pages 490–497, Springer-Verlag, 1992.
8. M. Girault. Low-size Coupons for Low-cost IC Cards. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, *Proceedings of Cardis 2000*, IFIP Conference Proceedings 180, pages 39–50, Kluwer Academic Publishers, 2000.
9. M. Girault and D. Lefranc. Public Key Authentication With One (On-line) Single Addition. In M. Joye and J.J. Quisquater, editors, *Proceedings of CHES '04*, LNCS 3156, pages 413–427, Springer-Verlag, 2004.
10. M. Girault and J. Stern. On the Length of Cryptographic Hash-values Used in Identification Schemes. In Y. Desmedt, editor, *Proceedings of Crypto '94*, LNCS 839, pages 202–215, Springer-Verlag, 1994.
11. T. Good, W. Chelton, and M. Benaissa. Review of Stream Cipher Candidates From a Low Resource Hardware Perspective. Available via `http://www.ecrypt.eu.org/`.
12. ISO/IEC. International Standard ISO/IEC 9798 Part 5: Mechanisms Using Zero-knowledge Techniques. December, 2004
13. IST-1999-12324. Final Report of European Project IST-1999-12324: New European Schemes for Signatures, Integrity, and Encryption (NESSIE). Available via `https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf`.
14. A. Juels and S. Weis. Authenticating Pervasive Devices with Human Protocols. In V. Shoup, editor, *Proceedings of Crypto 05*, LNCS 3126, 293-198, Springer-Verlag, 2005.
15. A. Menezes, P.C. van Oorschot, and S. Vanstone. The Handbook of Applied Cryptography. CRC Press, 1996.
16. G.E. Moore. Cramming More Components Onto Integrated Circuits. Electronics, April 19, 1965. Available via `www.intel.com`.
17. National Institute of Standards and Technology. FIPS 197: Advanced Encryption Standard. Available via `http://csrc.nist.gov/publications/fips/`.
18. B. Parhami. Computer Arithmetic: Algorithms and Hardware Designs. Oxford University Press, 2000.
19. G. Poupard and J. Stern. Secuity Analysis of a Practical "On the Fly" Authentication and Signature Generation. In K. Nyberg, editor, *Proceedings of Eurocrypt '98*, LNCS 1403, pages 422–436, Springer-Verlag, 1998.
20. S. Weis. Security and Privacy in Radio-Frequency Identification Devices. M.Sc. Thesis. May 2003.

A Bit-Slice Implementation of the Whirlpool Hash Function*

Karl Scheibelhofer

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria
karl.scheibelhofer@iaik.tugraz.at

Abstract. This work presents a bit-slice implementation of the Whirlpool hash function for 64-bit CPUs, which processes a single input block in one pass. It describes the general approach for developing the formulas and presents the results. This implementation does not need table lookups that depend on the data, which makes it immune against cache timing attacks, e.g. if used in an HMAC. Moreover, it requires 63% less memory (code and data) than the reference implementation of Whirlpool, and the performance of an implementation in C that uses some SSE2 instructions is only about 40% less. Additional improvements seem possible.

1 Introduction

Recently, cryptanalysis of hash functions has attracted attention in the crypto community. Theoretical and practical attacks were published (e.g. [10]) that apply to widely used hash functions like MD5 and SHA-1. These new attacks triggered discussions about successors for these algorithms. One of them is the Whirlpool hash function, which was created by Paulo Barreto and Vincent Rijmen. [2] specifies the version we refer to in this document. Recently, it was adopted as an ISO standard in ISO/IEC 10118-3 [1].

In addition to the cryptographic requirements for hash functions, a hash function should be efficient to compute. The most important performance aspects are data throughput, code size and memory footprint. However, there are also more subtle facets like cache access behavior. Typically, a developer has different options for the implementation of an algorithm. A very common situation is that we can trade off memory for performance to some extent. This means, if we use more memory we can reduce the number of computation steps and increase the throughput. The additional memory can, for instance, hold pre-computed values. A typical example is the S-box operation. Whirlpool contains an S-box operation, which can be implemented in an algorithmic way or using a table with pre-computed values.

* The work in this paper has been supported by the Austrian Science Fund (FWF), project P18138.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 385–401, 2007.

Sometimes, there are completely different approaches for implementing an algorithm. Biham [4] presented a new way to implement the DES block cipher. This so-called bit-slicing implementation performs the calculation in a bit-parallel manner. Simplified, we can say that we describe each bit of the result as a Boolean function of the input bits. Often, this is not that easy and straightforward as it sounds, at least if we want to get an implementation with an acceptable performance. [9] and [6] present bit-slice implementations of the Rijndael and AES block cipher that are suitable for 64-bit CPUs. Bit-slicing has the additional advantage that it does not need table-lookups that depend on the processed data. This makes it less susceptible to cache-timing attacks as described in [3].

In this paper, we describe a bit-slice implementation of the Whirlpool hash function that is suitable for CPUs with 64-bit registers and logical operations. This solution can process one 512-bit input block at a time. Due to its operation mode, it is impossible to process more blocks in parallel, which is what bit-slice implementations of block ciphers often do. Concurrent handling of multiple input blocks only works if the processing of one block does not need the result of the previous block, but generally, hash functions have such dependencies.

First, we give an introduction to Whirlpool. In the following section, we develop a bit-slice implementation step by step. We aim at a solution to group bits in 64-bit variables in a way that allows parallel processing. The resulting formulas will use 64-bit variables and operations as if they were 64 1-bit parallel. Thereafter, we discuss optimization issues for this approach and continue with a comparison to a reference implementation. Our implementation is written in C and uses SSE2 operations and x64-assembly for certain operations. The code only contains operations with constant execution time and uses not data-dependent table lookups, which makes is immune to timing attacks. The document closes with a summary and an outlook to further work.

2 Components of Whirlpool

For this work, only the compression function is of interest. We leave aside the padding and splitting of the message into blocks because these operations are common practice. Most of the formulas in this section were taken from [2].

Whirlpool is based on a dedicated 512-bit block cipher W and iterates the Miyaguchi-Preneel hashing scheme. The formulas for processing the blocks m_l $(1 \leq l \leq t)$ of the message M are

$$\begin{aligned} \eta_l &= \mu(m_l), \\ H_0 &= \mu(IV), \\ H_l &= W[H_{l-1}](\eta_l) \oplus H_{l-1} \oplus \eta_l, \qquad 1 \leq l \leq t \\ \textsc{Whirlpool}(M) &\equiv \mu^{-1}(H_t) \end{aligned}$$

Here, IV is the initialization vector, which contains 512-bit zero bits. The function μ fills an eight by eight matrix row by row with the octets of the

block of input data. μ^{-1} is its inverse. Whirlpool is specified by operations on this matrix. Each entry of the matrix is considered an element of $GF(2^8)$ with the reduction polynomial $p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$.

$$\mu : GF(2^8)^{64} \rightarrow M_{8\times 8}[GF(2^8)]$$

$$\mu(a) = b \Leftrightarrow b_{ij} = a_{8i+j}, \qquad 0 \leq i, j \leq 7$$

The dedicated block cipher W takes a key as parameter and is defined as

$$W[K] = (\overset{10}{\underset{r=1}{\bigcirc}} \rho[K^r]) \circ \sigma[K^0]$$

In this formula, K^0 and K^r are the round keys and $\rho[k]$ is the round function, which is iterated 10 times. The keys, the input and the output are $M_{8\times 8}[GF(2^8)]$ matrices. The key addition function $\sigma[k]$ is an XOR operation of the key k and the input. The round function is the composite mapping

$$\rho[k] \equiv \sigma[k] \circ \theta \circ \pi \circ \gamma$$

The function $\rho[k]$ and its components $\sigma[k]$, θ, π and γ are mappings of type

$$M_{8\times 8}[GF(2^8)] \rightarrow M_{8\times 8}[GF(2^8)]$$

The function γ is the non-linear layer and is defined using a substitution box (S-box) S, which contains three smaller components E, E^{-1} and R (see Fig. 1 and [2]).

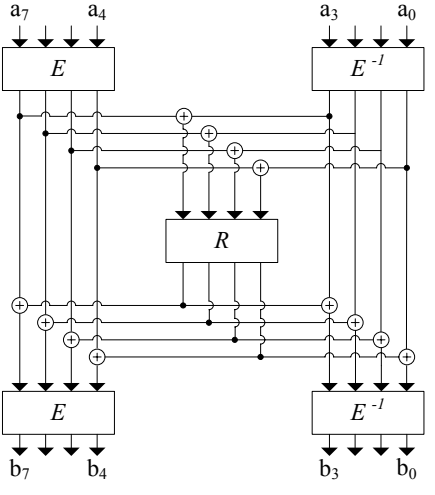

Fig. 1. The S-box of the Whirlpool Hash Function [8]

The function π is the cyclical permutation function. It rotates each column by a different offset, i.e. it leaves column zero as it is, rotates down column one by one position, column two by two positions and so on.

$$\pi(a) = b \Leftrightarrow b_{ij} = a_{(i-j) \bmod 8, j}, \qquad 0 \leq i, j \leq 7$$

The linear diffusion layer θ multiplies the input a by the constant matrix C.

$$\theta(a) = b \Leftrightarrow b = a \cdot C$$

The round keys K^r are calculated using round constants c^r and are defined as

$$\begin{aligned} K^0 &= K \\ K^r &= \rho[c^r](K^{r-1}) && r > 0 \\ c^r_{0j} &\equiv S[8(r-1)+j] && 0 \le j \le 7 \\ c^r_{ij} &\equiv 0 && 1 \le i \le 7, \quad 0 \le j \le 7 \end{aligned}$$

Fig. 2 shows that there are two block ciphers running in parallel, one for the data path and one for the key schedule.

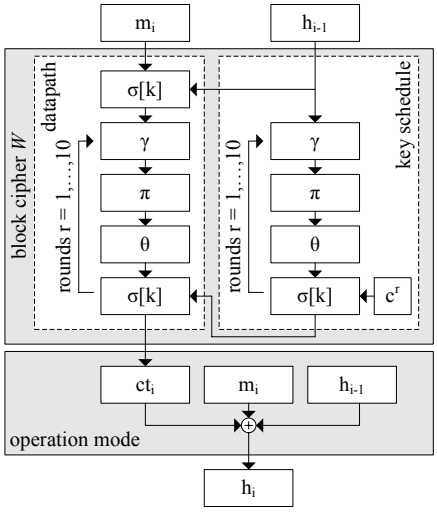

Fig. 2. Structure of Whirlpool [8]

3 The General Approach

The general approach is to change the data domain from $M_{8\times8}[GF(2^8)]$ to a different domain which is better suitable for a bit-parallel computation. Having chosen a domain, we transform the operations that comprise Whirlpool into that domain. We will mark the bit-sliced versions of the functions with a tilde. The functions θ, π and γ are the most important of them. In addition, we need functions that transform the input data into our new domain and the result back from this domain. Naturally, we adapt the input and output functions μ and μ^{-1} to do the mapping. We select $M_{64\times8}[GF(2)]$ as our new domain instead of $M_{8\times8}[GF(2^8)]$ in the original formulas, i.e. eight vectors, each with 64 bit. This selection allows fully exploiting 64-bit registers and operations of modern CPUs.

In addition to $\tilde{\mu}$ and $\tilde{\mu}^{-1}$, we get a bit-sliced version of the round function $\rho[k]$ called $\tilde{\rho}[k]$

$$\tilde{\rho}[k] \equiv \tilde{\sigma}[k] \circ \tilde{\theta} \circ \tilde{\pi} \circ \tilde{\gamma}$$

The following subsections describe the components.

3.1 The Input Function $\tilde{\mu}$ and the Output Function $\tilde{\mu}^{-1}$

Instead of mapping $GF(2^8)^{64}$ to $M_{8\times 8}[GF(2^8)]$, we define the input function $\tilde{\mu}$ to[1]

$$\tilde{\mu} : GF(2^8)^{64} \rightarrow M_{64\times 8}[GF(2)]$$

$$\begin{aligned}\tilde{\mu}(a) = b \Leftrightarrow b_i = [&a_0|_i, a_8|_i, a_{16}|_i, \ldots, a_{56}|_i,\\ &a_1|_i, a_9|_i, a_{17}|_i, \ldots, a_{57}|_i,\\ &\vdots\\ &a_7|_i, a_{15}|_i, a_{23}|_i, \ldots, a_{63}|_i],\ 0 \leq i \leq 7\end{aligned}$$

This function traverses the octets column by column if we consider the input an eight by eight matrix as in the traditional description. It collects the bit at index zero of each octet (i.e. an element of $GF(2^8)$) in a single 64-bit vector. It repeats that for all eight bits. This is the reason why it is called bit-slice, because we slice the input bit-wise and not byte-wise.

The inverse $\tilde{\mu}^{-1}$ takes the eight 64-bit vectors and converts it back into 64 octets.

3.2 The Non-linear Layer $\tilde{\gamma}$

This function performs the S-box operations. For getting a bit-slice implementation of the S-box, we need a Boolean formula for each output bit of the S-box in terms of the eight input bits. [2, Appendix B] already presents suitable formulas for the components E, E^{-1} and R of the S-box.[2] We only need some additional XOR operations to combine the results (see Fig. 1).

The formulas allow the computation of the eight bits of a single S-box operation. The S-box and thus the formulas are the same for all 64 octets. Thus, we can use the same formulas with the 64-bit vectors instead of just using single-bit variables. We use the 64-bit vectors and operations as if we had 64 parallel 1-bit CPUs with 1-bit registers. This works because all required operations (like logical AND, OR, XOR and NOT) only affect corresponding bits in the operands, e.g. if we XOR two 64-bit values, bit 5 of the first operand is XORed with bit 5 of the second operand. The values of all other bits have no influence for bit 5 of the result.

[1] The notation $a|_i$ means the i-th bit of element a.

[2] Shorter formulas for the components may exist than those given in [2, Appendix B]. However, finding optimal formulas is a complex problem.

3.3 The Cyclical Permutation $\tilde{\pi}$

In $M_{8\times 8}[GF(2^8)]$, π is just a rotation of the values of each column by a fixed number of positions. Since the first column remains untouched, we need to load and store 56 octets. Here, we have the bits collected in 64-bit vectors column by column. This means, the first octet of a vector contains the eight bits of the first column, the second octet the eight bits of the second column and so on. In consequence, we can leave the first octet as it is because the first column needs no rotation. We have to rotate the second octet by one bit, the third octet by two bits and so on. This repeats for all eight 64-bit vectors.

3.4 The Linear Diffusion Layer $\tilde{\theta}$

Developing a bit-slice version of the function θ is more work than for the other functions because there is no prior work that we can build upon. We start by investigating the linear diffusion layer of the original Whirlpool specification to derive formulas for the single bits of the result of this function. Then we will develop versions of these formulas that lend themselves to bit-slicing. On the first glance, this function looks very complicated to adapt for our purpose, but it turns out that it can be computed efficiently. Written out, the θ function is

$$\begin{bmatrix} b_{00} & b_{01} & \dots & b_{07} \\ b_{10} & b_{11} & \dots & b_{17} \\ \vdots & \vdots & & \vdots \\ b_{70} & b_{71} & \dots & b_{77} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & \dots & a_{07} \\ a_{10} & a_{11} & \dots & a_{17} \\ \vdots & \vdots & & \vdots \\ a_{70} & a_{71} & \dots & a_{77} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 4 & 1 & 8 & 5 & 2 & 9 \\ 9 & 1 & 1 & 4 & 1 & 8 & 5 & 2 \\ 2 & 9 & 1 & 1 & 4 & 1 & 8 & 5 \\ 5 & 2 & 9 & 1 & 1 & 4 & 1 & 8 \\ 8 & 5 & 2 & 9 & 1 & 1 & 4 & 1 \\ 1 & 8 & 5 & 2 & 9 & 1 & 1 & 4 \\ 4 & 1 & 8 & 5 & 2 & 9 & 1 & 1 \\ 1 & 4 & 1 & 8 & 5 & 2 & 9 & 1 \end{bmatrix}$$

It is noteworthy that the columns and rows of the constant matrix C are rotations of each other. Multiplication of an element by a constant means a polynomial multiplication modulo the reduction polynomial $p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$. Formulas for the products are given in Appendix A.1.

Note that the constant multipliers only rotate to down from one column to the next. Instead of rotating the multipliers to down, we can also rotate the operands to the left, such that operands with the same multipliers are aligned. This leads to the following formula (indices are mod 8).[3]

$$b_{ij} = a_{ij} \oplus 9a_{i(j+1)} \oplus 2a_{i(j+2)} \oplus 5a_{i(j+3)} \oplus 8a_{i(j+4)} \oplus a_{i(j+5)} \oplus 4a_{i(j+6)} \oplus a_{i(j+7)}$$

We did not change the formulas; we only reordered the terms. Now, we have effectively the same formula for all rows and all columns. This is what we need for parallelization.

We have the input matrix in a bit-sliced form, all bits with index 0 in one 64-bit word. We start with the 0-bit of a_{00} and proceed column by column, i.e.

[3] All formulas are given in Appendix A.2.

continue with $a_{10},\ldots,\ a_{70}$, then $a_{01},\ a_{11},\ldots,\ a_{71}$, and up to $a_{07},\ a_{17},\ldots,\ a_{77}$. For the first term, we already have the columns in the right position. For the other terms, we need some preparation. The second term starts one column to the right of the result column (shown by the $j+1$ index). To get the bits in the registers in the corresponding position for an XOR operation, we have to rotate 8 bits to the left the register that holds the required input bits.

Performing the multiplications with the constants leads to formulas for the individual bits of the output. For example, for $b_{i0}|_0$, we get[4]

$$b_{i0}|_0 = a_{i0}|_0 \oplus (a_{i1}|_0 \oplus a_{i1}|_5) \oplus a_{i2}|_7 \oplus (a_{i3}|_0 \oplus a_{i3}|_6) \oplus a_{i4}|_5 \oplus a_{i5}|_0 \\ \oplus a_{i6}|_6 \oplus a_{i7}|_0$$

After generalizing these formulas to apply to all eight columns, we get[5]

$$b_{ij}|_0 = a_{ij}|_0 \oplus (a_{i(j+1)}|_0 \oplus a_{i(j+1)}|_5) \oplus a_{i(j+2)}|_7 \oplus (a_{i(j+3)}|_0 \oplus a_{i(j+3)}|_6) \\ \oplus a_{i(j+4)}|_5 \oplus a_{i(j+5)}|_0 \oplus a_{i(j+6)}|_6 \oplus a_{i(j+7)}|_0$$

This formula shows that we can calculate all bits with index 0 of all 64 output-bytes in parallel. This was our goal. Implementing these formulas is straightforward if we interpret $a_{i(j+k)}|_b$ as taking the variable that holds all bits with index b and rotating it to the left by eight times k bits.

Let as_b be the 64-bit variable that holds all bits (from all 64 input octets a_{ij}) with bit index b.

$$as_b = [a_{ij}|_b], \qquad 0 \le i, j \le 7$$

We assume that the bits are collected per column as shown for $\tilde{\mu}$. To evaluate the values of bs_b, we calculate $rl^{8.k}(as_b)$ to get the value of an expression of the form $a_{i(j+k)}|_b$, where $rl^n(as_b)$ is a rotation to the left of as_b by n bits. For instance, the first output value is[6]

$$bs_0 = as_0 \oplus rl^8(as_0) \oplus rl^8(as_5) \oplus rl^{16}(as_7) \oplus rl^{24}(as_0) \oplus rl^{24}(as_6) \\ \oplus rl^{32}(as_5) \oplus rl^{40}(as_0) \oplus rl^{48}(as_6) \oplus rl^{56}(as_0)$$

3.5 The Key Addition $\tilde{\sigma}[\tilde{k}]$

The key addition function $\tilde{\sigma}$ is still a simple XOR operation. Thus, there are only eight XOR operations with 64-bit operands. However, we have to be aware of the fact that the key and the operand are in the bit-slice domain.

3.6 The Round Constants

The round constants are used as keys for the dedicated block cipher in the key schedule. Our block cipher is bit-sliced, and thus, we also need the keys in bit-slice representation. In general, we have two options. First, we can compute the

[4] All formulas are given in Appendix A.3.
[5] All formulas are given in Appendix A.4.
[6] All formulas are given in Appendix A.5.

round constants using the function $\tilde{\gamma}$ (maybe even every time we need them) or second, we can compute them once and hard-code them.

A simple way for computing the round constants using $\tilde{\gamma}$ is the following. We start with a $M_{8\times8}[GF(2^8)]$ matrix. We will bit-slice this matrix using $\tilde{\mu}$ and then apply $\tilde{\gamma}$ to get a bit-sliced round constant. (If we calculated a round constant in the $M_{8\times8}[GF(2^8)]$ domain using γ, we would also setup a $M_{8\times8}[GF(2^8)]$ matrix.) For the first round constant, we would initialize the first row with the values $0, 1, 2, 3, 4, 5, 6, 7$. The second to the eighth row need a value which gets zero after applying γ. `0x81` hexadecimal is the value that maps to zero, and thus, we set all elements of the rows two to eight to the value `0x81`. Then, we can apply γ to the matrix. The rows two to eight will contain just zeros thereafter. The first row would hold the results of the S-box operations. In bit-slicing mode, we can do the same. The only difference is that we need to convert the input matrix (with the same values as before) into the bit-slice domain using the function $\tilde{\mu}$. Then we apply function $\tilde{\gamma}$ and get the first round constant in the bit-slice domain. For the other round constants, the procedure is the same, only the values in the first row of the input matrix change. For the second round constant, the first row would hold the values $8, 9, 10, 11, 12, 13, 14, 15$, for the third, it would hold $16, 17, 18, 19, 20, 21, 22, 23$ and so on.

4 Optimization Issues

In this section, we discuss some optimization issues of the bit-slice implementation. This includes hints for improving performance and discussion of different implementation options.

4.1 The Input Function $\tilde{\mu}$

Collecting the individual bits of the $GF(2^8)$ octets in 64-bit vectors can be rather expensive if we have only logical instructions. In this case, we have to copy the bits from the octets into the 64-bit vectors, using shift and mask operations. To get the bits out of the octet, we use an AND operation, and to insert it into the vector, we use an OR operation (an XOR would also work). This requires nearly 2000 operations (shift, AND and OR) to process a 512 bit input block.[7]

New CPUs (e.g. Intel Pentium 4™) often support Single Instruction Multiple Data (SIMD) instructions sets like SSE2. SSE2 contains the PMOVMSKB instruction, and using it can reduce the required number of operations for $\tilde{\mu}$ to 120, and tests showed an actual throughput increase of over 80%. Section 5 presents an improved bit-slice implementation that uses this instruction.

4.2 The Non-linear Layer $\tilde{\gamma}$

We can implement the Boolean formulas of [2, Appendix B] to get the components E, E^{-1} and R. Optimization of $\tilde{\gamma}$ would required shorter formulas, however, finding optimal formulas is non-trivial.

[7] A faster version is possible using the SWAPMOVE technique. See section 7.

4.3 The Cyclical Permutation $\tilde{\pi}$

In the operation domain of the normal Whirlpool specification, the function π is rather simple. It is easy to describe and efficient to implement. Unfortunately, in bit-slice mode, this is not the case.

We need to rotate the octets of each 64-bit vector individually. Notice that this is not the same as a rotation of the complete 64-bit vector. In our case, we have to treat the eight octets of a vector as isolated values. Usually, we would have to extract each octet, rotate it as an octet value and then move it back to its original position.

Arranging the bits in the 64-bit vectors row by row instead of column by column (i.e. modifying the function $\tilde{\mu}$) makes the situation even worse. In this case, we would need to move around single bits instead of octets, which is even more costly.

Optimizing this function is essential because it takes 344 operations within the round function $\tilde{\rho}$ (see Section 5). Using Assembler instructions of an AMD-64 CPU, one can use a combination of rotations of a 64-bit register and rotations of the 8-bit sub-register, e.g. rotating registers RAX and AL alternately. This way, this function needs only 88 instructions.

4.4 The Linear Diffusion Layer $\tilde{\theta}$

The formulas for the function $\tilde{\theta}$ contain only rotations and XOR operations, and the number of operations is comparable to the function $\tilde{\gamma}$ (S-boxes). Most CPUs process such instructions very efficiently. However, there may still be room for optimizations. For example, we can exploit the distributivity property of the rotation operation.

$$rl^8(as_0) \oplus rl^8(as_5) = rl^8(as_0 \oplus as_5)$$

This saves a rotation operation. In addition, we can calculate certain terms once and reuse the result.

On some systems, it may also be feasible to implement the rotations by loading a register with a suitable offset. If we write the value into the memory two times, one right after the other, we can get the rotated value by loading a register from this memory location with an offset. To get a rotation of eight bit to the left, we load the register with the base address plus one byte offset. This works because we only need rotations that are multiples of eight bits. The efficiency of this approach depends on the performance of the CPU for non-aligned memory access.

4.5 The Key Addition $\tilde{\sigma}[\tilde{k}]$

The key addition is a simple XOR operation for each of the eight 64-bit vectors. This is negligible in contrast to the other functions of the block cipher. Moreover, there is not much space for improvements in this function.

4.6 The Round Constants

If we compute the round constants, the implementation is straightforward from the description above. If we use pre-computed round constants and hard-code them, we can use them directly. This takes eight constants for each of the ten rounds with 64-bit each, this sums to 640 byte. In the non-bit-sliced Whirlpool, we only have non-zero values in the first row of each round constant. Thus, we can store all non-zero values within 80 byte. This seems to be an advantage over the bit-sliced version. The constants for the bit-sliced version have a hamming weight of at most eight. In addition, all bits of an octet are constantly zero except the most significant bit. If we need an implementation with the smallest possible memory demand, we can consider storing these constants in a compressed form.

5 Bit-Slicing vs. Non-bit-Slicing

In this section, we compare a table-based implementation of the Whirlpool hashing function with two bit-slice implementations. The table-based implementation is the reference implementation in C by Paulo Barreto and Vincent Rijmen. It does not simply execute the formulas of the specification in C; it is optimized for maximum speed. Popular crypto libraries like OpenSSL[8] and Crypto++[9] use this code, and this approach does not seem to offer significant potential for further optimization. The first bit-sliced implementation is written in pure C, and the second is written in C with the $\tilde{\mu}$ function using compiler intrinsics to access SSE2 instructions and $\tilde{\pi}$ implemented in assembly language (both versions developed by the author of this paper). We used the Microsoft Visual C++ 2005 compiler for 64-bit platforms on a Windows 2003 Server x64 Edition with an AMD Opteron 146 processor. The part that we analyze is the compression function of Whirlpool. Please note that we do not consider memory access operations and overhead of function calls when we count the instructions. [10]

Table 1 and 2 show that the simple bit-slice implementation needs 3.3 times more operations than the table-based operation. The input preparation already takes 1984 operations.

Table 2 shows the function $\tilde{\rho}[\tilde{k}]$ of the simple bit-sliced implementation. Note, that the block cipher $\tilde{W}[\tilde{K}]$ applies $\tilde{\rho}[\tilde{k}]$ 20 times, 10 times for the key schedule and 10 times for the data path. Here, we can see that the cyclical permutation $\tilde{\pi}$ accounts for 344 operations out of the total of 498, which is 69%. This operation is costly if implemented in C because it needs to rotate individual byte-blocks inside of 64-bit variables.

[8] Latest version of OpenSSL as of 20 September 2006 in the CVS repository of the project.

[9] Version 5.2.1.

[10] This provides realistic results. Modern CPUs usually have separate execution units for memory access, which can operate in parallel to the ALU. If the number of memory access operations is lower than the number of logic operations, they may not show at all in the overall performance figures. This heavily depends on the internal processor architecture (see [5] for further details).

Table 1. Instruction counts for the *reference* implementation

Step	AND	XOR	SH	Sum
μ	56	56	56	168
Apply K^0		8		8
Cipher(10 R)	1120	1200	1120	3440
Miyag.-Pren.		16		16
Total				3632

Table 2. Instruction counts for the *simple* bit-slice implementation

Step	AND	XOR	SH	OR	NOT	ROT	Sum
$\tilde{\gamma}$	8	39		13	5		65
$\tilde{\pi}$	120		112	112			344
$\tilde{\theta}$		25				56	81
$\tilde{\sigma}$		8					8
Total $\tilde{\rho}[\tilde{k}]$							498
$\tilde{\mu}$	512		960	512			1984
Apply $\tilde{K}^0$		8					8
Cipher(10 R)	2560	1440	2240	2500	100	1120	9960
Miyag.-Pren.		16					16
Total							11968

The improved bit-slice implementation performs better. This implementation has an optimized implementation of the function $\tilde{\mu}$ for bit-slicing the input, which uses SSE2 instructions like PMOVMSKB and rotations. Moreover, it includes an assembly implementation of the function $\tilde{\pi}$, which uses the combined 64-bit/8-bit rotations as described in 4.3. This implementation requires only 88 rotation instructions (56 8-bit rotations and 32 64-bit rotations).

Table 3 shows that the number of operations for the input preparation reduced to 120, and the operations for the 20 rounds of the block cipher reduced to about the half. In total, this implementation requires only 37% more instructions than the table-based version. The actual performance may vary because different processors execute instructions with a different speed. With the RDTSC instruction we measured the required clock cycles on an AMD Opteron 146 CPU, which took 2881 cycles for processing a block with the reference implementation and 4541 cycles for the improved bit-slice variant. This is a ratio of 1.58.

A big advantage of the bit-sliced implementations is that they need less memory, as Table 4 shows, because they do not need any lookup tables for operations like the S-boxes. In fact, the bit-slice code does not include any operation whose execution time depends on the processed data. No data is used as a branch condition, as an index or as bit-count in a rotation operation. These are the best premises for countering timing attacks. Excluding tables and constants, the pure code sizes are similar. The table-based implementation takes 5194 byte versus 7320 byte for the improved bit-slice version. Thus, the total memory

Table 3. Instruction counts for the *improved* bit-slice implementation

Step	AND	XOR	OR	NOT	ROT	SSE2	Sum
$\tilde{\gamma}$	8	39	13	5			65
$\tilde{\pi}$					88		88
$\tilde{\theta}$		25			56		81
$\tilde{\sigma}$		8					8
Total $\tilde{\rho}[\tilde{k}]$							242
$\tilde{\mu}$						120	120
Apply $\tilde{K}^0$		8					8
Cipher(10 R)	160	1440	260	100	2880		4840
Miyag.-Pren.		16					16
Total							4984

requirements are 8216 byte for the improved bit-slice version in contrast to 21914 byte for the reference variant, which is 63% less. The required memory for the round constants of the bit-slice variant can be reduced further by storing the constants in a compressed form (see 4.6).

Table 4. Memory requirements (in byte) of the reference and the *improved* bit-slice implementation

		Table-Based	Bit-Sliced
Constants	Round Constants	80	640
	Lookup Tables	16384	0
During Compression	State	64	64
	Input Data	64	64
	Key	64	64
	Working Variable	64	64
Sum of Data		16720	896
Code Size		5194	7320
Total Memory		21914	8216

6 Conclusion

This work presented a bit-sliced version of Whirlpool that can be used as a drop-in replacement for existing implementations. Even though our improved bit-slice implementation shown here needs 37% more operations than the reference implementation that uses large tables, this work shows that such an alternative implementation has some benefits. It needs significantly less memory, about 95% less data memory and 63% less total memory (code and data) compared to the aforementioned reference implementation. In addition, it contains no instructions whose execution time depends on the processed data, like data-dependent table lookups. This prevents susceptibility to cache timing attacks, which can be an issue in certain applications.

7 Further Work

Modern PC processors like Intel Pentium 4 or AMD Opteron support the SSE2 extension, which offers 128-bit registers and instructions. This instruction set contains all logical operations required to implement the bit-sliced Whirlpool. An implementation that operates with SSE2 registers only would also run on 32-bit CPUs that support SSE2, which are more common than 64-bit CPUs. A pure SSE2 implementation would offer an additional advantage due to the wider registers. It would allow processing the data path and the key schedule in parallel (see Fig. 2). This can result in a big performance gain. However, the actual results may vary depending on the execution speed of SSE2 operations compared to operations on general-purpose registers.

Because the $\tilde{\mu}$ function is actually a bit-permutation operation, it is possible to implement this operation using the SWAPMOVE technique that is also used for permutations in several DES implementations (e.g. in OpenSSL and [7]). A first version required 24 shift, 36 XOR and 12 AND operations and reversing the byte-order of the result, which can be implemented using 8 BSWAP operations. However, this speed-up had little influence on the speed of the complete compression function.

Acknowledgements

We want to thank Vincent Rijmen for his support in this work. The discussions with him and his comments were always valuable.

References

1. *ISO/IEC 10118-3:2004: Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions.* International Organization for Standardization, Geneva, Switzerland, 2004.
2. Paulo S.L.M. Barreto and Vincent Rijmen. The WHIRLPOOL Hashing Function, May 2003.
3. Daniel J. Bernstein. Cache-timing attacks on AES, April 2005.
4. Eli Biham. A Fast New DES Implementation in Software. In Eli Biham, editor, *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.
5. Agner Fog. The microarchitecture of Intel and AMD CPU's, August 2006.
6. Mitsuru Matsui. How Far Can We Go on the x64 Processors? In *Fast Software Encryption*, March 2006.
7. Lauren May, Lyta Penna, and Andrew Clark. An implementation of bitsliced des on the pentium mmx$^{\text{tm}}$ processor. In Ed Dawson, Andrew Clark, and Colin Boyd, editors, *ACISP*, volume 1841 of *Lecture Notes in Computer Science*, pages 112–122. Springer, 2000.
8. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. A compact FPGA implementation of the hash function whirlpool. In Steven J. E. Wilton and André DeHon, editors, *FPGA*, pages 159–166. ACM, 2006.

9. Atri Rudra, Pradeep K. Dubey, Charanjit S. Jutla, Vijay Kumar, Josyula R. Rao, and Pankaj Rohatgi. Efficient rijndael encryption implementation with composite field arithmetic. In *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pages 171–184, London, UK, 2001. Springer-Verlag.
10. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

A Formulas for Function $\tilde{\theta}$

A.1 Products

The multiplications with the relevant constants are in Table 5. Each row element shows the result of a single bit as a logical combination of bits of the input a.

Table 5. Formulas for products

Result Bit	Constant Multiplier					
	1	2	4	5	8	9
0	$a\|_0$	$a\|_7$	$a\|_6$	$a\|_0 \oplus a\|_6$	$a\|_5$	$a\|_0 \oplus a\|_5$
1	$a\|_1$	$a\|_0$	$a\|_7$	$a\|_1 \oplus a\|_7$	$a\|_6$	$a\|_1 \oplus a\|_6$
2	$a\|_2$	$a\|_1 \oplus a\|_7$	$a\|_0 \oplus a\|_6$	$a\|_0 \oplus a\|_2 \oplus a\|_6$	$a\|_5 \oplus a\|_7$	$a\|_2 \oplus a\|_5 \oplus a\|_7$
3	$a\|_3$	$a\|_2 \oplus a\|_7$	$a\|_1 \oplus a\|_6 \oplus a\|_7$	$a\|_1 \oplus a\|_3 \oplus a\|_6 \oplus a\|_7$	$a\|_0 \oplus a\|_5 \oplus a\|_6$	$a\|_0 \oplus a\|_3 \oplus a\|_5 \oplus a\|_6$
4	$a\|_4$	$a\|_3 \oplus a\|_7$	$a\|_2 \oplus a\|_6 \oplus a\|_7$	$a\|_2 \oplus a\|_4 \oplus a\|_6 \oplus a\|_7$	$a\|_1 \oplus a\|_5 \oplus a\|_6 \oplus a\|_7$	$a\|_1 \oplus a\|_4 \oplus a\|_5 \oplus a\|_6 \oplus a\|_7$
5	$a\|_5$	$a\|_4$	$a\|_3 \oplus a\|_7$	$a\|_3 \oplus a\|_5 \oplus a\|_7$	$a\|_2 \oplus a\|_6 \oplus a\|_7$	$a\|_2 \oplus a\|_5 \oplus a\|_6 \oplus a\|_7$
6	$a\|_6$	$a\|_5$	$a\|_4$	$a\|_4 \oplus a\|_6$	$a\|_3 \oplus a\|_7$	$a\|_3 \oplus a\|_6 \oplus a\|_7$
7	$a\|_7$	$a\|_6$	$a\|_5$	$a\|_5 \oplus a\|_7$	$a\|_4$	$a\|_4 \oplus a\|_7$

A.2 Formulas for Column 0 in the Result

The elements b_{i0} to b_{i7} $(0 \leq i \leq 7)$ in a result column are calculated as follows:

$$
\begin{aligned}
b_{i0} &= a_{i0} \oplus 9a_{i1} \oplus 2a_{i2} \oplus 5a_{i3} \oplus 8a_{i4} \oplus a_{i5} \oplus 4a_{i6} \oplus a_{i7} \\
b_{i1} &= a_{i0} \oplus a_{i1} \oplus 9a_{i2} \oplus 2a_{i3} \oplus 5a_{i4} \oplus 8a_{i5} \oplus a_{i6} \oplus 4a_{i7} \\
b_{i2} &= 4a_{i0} \oplus a_{i1} \oplus a_{i2} \oplus 9a_{i3} \oplus 2a_{i4} \oplus 5a_{i5} \oplus 8a_{i6} \oplus a_{i7} \\
b_{i3} &= a_{i0} \oplus 4a_{i1} \oplus a_{i2} \oplus a_{i3} \oplus 9a_{i4} \oplus 2a_{i5} \oplus 5a_{i6} \oplus 8a_{i7} \\
b_{i4} &= 8a_{i0} \oplus a_{i1} \oplus 4a_{i2} \oplus a_{i3} \oplus a_{i4} \oplus 9a_{i5} \oplus 2a_{i6} \oplus 5a_{i7} \\
b_{i5} &= 5a_{i0} \oplus 8a_{i1} \oplus a_{i2} \oplus 4a_{i3} \oplus a_{i4} \oplus a_{i5} \oplus 9a_{i6} \oplus 2a_{i7} \\
b_{i6} &= 2a_{i0} \oplus 5a_{i1} \oplus 8a_{i2} \oplus a_{i3} \oplus 4a_{i4} \oplus a_{i5} \oplus a_{i6} \oplus 9a_{i7} \\
b_{i7} &= 9a_{i0} \oplus 2a_{i1} \oplus 5a_{i2} \oplus 8a_{i3} \oplus a_{i4} \oplus 4a_{i5} \oplus a_{i6} \oplus a_{i7}
\end{aligned}
$$

A.3 Bits for Column 0 in the Result

The notation $b_{00}|_0$ denotes the bit at index 0 of element b_{00}. The individual bits of b_{i0} are as follows:

$$
\begin{aligned}
b_{i0}|_0 &= a_{i0}|_0 \oplus (a_{i1}|_0 \oplus a_{i1}|_5) \oplus a_{i2}|_7 \oplus (a_{i3}|_0 \oplus a_{i3}|_6) \oplus a_{i4}|_5 \oplus a_{i5}|_0 \\
&\quad \oplus a_{i6}|_6 \oplus a_{i7}|_0 \\
b_{i0}|_1 &= a_{i0}|_1 \oplus (a_{i1}|_1 \oplus a_{i1}|_6) \oplus a_{i2}|_0 \oplus (a_{i3}|_1 \oplus a_{i3}|_7) \oplus a_{i4}|_6 \oplus a_{i5}|_1 \\
&\quad \oplus a_{i6}|_7 \oplus a_{i7}|_1 \\
b_{i0}|_2 &= a_{i0}|_2 \oplus (a_{i1}|_2 \oplus a_{i1}|_5 \oplus a_{i1}|_7) \oplus (a_{i2}|_1 \oplus a_{i2}|_7) \\
&\quad \oplus (a_{i3}|_0 \oplus a_{i3}|_2 \oplus a_{i3}|_6) \oplus (a_{i4}|_5 \oplus a_{i4}|_7) \oplus a_{i5}|_2 \\
&\quad \oplus (a_{i6}|_0 \oplus a_{i6}|_6) \oplus a_{i7}|_2 \\
b_{i0}|_3 &= a_{i0}|_3 \oplus (a_{i1}|_0 \oplus a_{i1}|_3 \oplus a_{i1}|_5 \oplus a_{i1}|_6) \oplus (a_{i2}|_2 \oplus a_{i2}|_7) \\
&\quad \oplus (a_{i3}|_1 \oplus a_{i3}|_3 \oplus a_{i3}|_6 \oplus a_{i3}|_7) \oplus (a_{i4}|_0 \oplus a_{i4}|_5 \oplus a_{i4}|_6) \\
&\quad \oplus a_{i5}|_3 \oplus (a_{i6}|_1 \oplus a_{i6}|_6 \oplus a_{i6}|_7) \oplus a_{i7}|_3 \\
b_{i0}|_4 &= a_{i0}|_4 \oplus (a_{i1}|_1 \oplus a_{i1}|_4 \oplus a_{i1}|_5 \oplus a_{i1}|_6 \oplus a_{i1}|_7) \\
&\quad \oplus (a_{i2}|_3 \oplus a_{i2}|_7) \oplus (a_{i3}|_2 \oplus a_{i3}|_4 \oplus a_{i3}|_6 \oplus a_{i3}|_7) \\
&\quad \oplus (a_{i4}|_1 \oplus a_{i4}|_5 \oplus a_{i4}|_6 \oplus a_{i4}|_7) \oplus a_{i5}|_4 \\
&\quad \oplus (a_{i6}|_2 \oplus a_{i6}|_6 \oplus a_{i6}|_7) \oplus a_{i7}|_4 \\
b_{i0}|_5 &= a_{i0}|_5 \oplus (a_{i1}|_2 \oplus a_{i1}|_5 \oplus a_{i1}|_6 \oplus a_{i1}|_7) \oplus a_{i2}|_4 \\
&\quad \oplus (a_{i3}|_3 \oplus a_{i3}|_5 \oplus a_{i3}|_7) \oplus (a_{i4}|_2 \oplus a_{i4}|_6 \oplus a_{i4}|_7) \\
&\quad \oplus a_{i5}|_5 \oplus (a_{i6}|_3 \oplus a_{i6}|_7) \oplus a_{i7}|_5 \\
b_{i0}|_6 &= a_{i0}|_6 \oplus (a_{i1}|_3 \oplus a_{i1}|_6 \oplus a_{i1}|_7) \oplus a_{i2}|_5 \oplus (a_{i3}|_4 \oplus a_{i3}|_6) \\
&\quad \oplus (a_{i4}|_3 \oplus a_{i4}|_7) \oplus a_{i5}|_6 \oplus a_{i6}|_4 \oplus a_{i7}|_6 \\
b_{i0}|_7 &= a_{i0}|_7 \oplus (a_{i1}|_4 \oplus a_{i1}|_7) \oplus a_{i2}|_6 \oplus (a_{i3}|_5 \oplus a_{i3}|_7) \\
&\quad \oplus a_{i4}|_4 \oplus a_{i5}|_7 \oplus a_{i6}|_5 \oplus a_{i7}|_7
\end{aligned}
$$

A.4 Bits for All Elements in the Result

These are the generalized formulas the the bits of all result elements.

$$
\begin{aligned}
b_{ij}|_0 &= a_{ij}|_0 \oplus (a_{i(j+1)}|_0 \oplus a_{i(j+1)}|_5) \oplus a_{i(j+2)}|_7 \oplus (a_{i(j+3)}|_0 \oplus a_{i(j+3)}|_6) \\
&\quad \oplus a_{i(j+4)}|_5 \oplus a_{i(j+5)}|_0 \oplus a_{i(j+6)}|_6 \oplus a_{i(j+7)}|_0 \\
b_{ij}|_1 &= a_{ij}|_1 \oplus (a_{i(j+1)}|_1 \oplus a_{i(j+1)}|_6) \oplus a_{i(j+2)}|_0 \oplus (a_{i(j+3)}|_1 \oplus a_{i(j+3)}|_7) \\
&\quad \oplus a_{i(j+4)}|_6 \oplus a_{i(j+5)}|_1 \oplus a_{i(j+6)}|_7 \oplus a_{i(j+7)}|_1 \\
b_{ij}|_2 &= a_{ij}|_2 \oplus (a_{i(j+1)}|_2 \oplus a_{i(j+1)}|_5 \oplus a_{i(j+1)}|_7) \oplus (a_{i(j+2)}|_1 \oplus a_{i(j+2)}|_7) \\
&\quad \oplus (a_{i(j+3)}|_0 \oplus a_{i(j+3)}|_2 \oplus a_{i(j+3)}|_6) \oplus (a_{i(j+4)}|_5 \oplus a_{i(j+4)}|_7) \\
&\quad \oplus a_{i(j+5)}|_2 \oplus (a_{i(j+6)}|_0 \oplus a_{i(j+6)}|_6) \oplus a_{i(j+7)}|_2 \\
b_{ij}|_3 &= a_{ij}|_3 \oplus (a_{i(j+1)}|_0 \oplus a_{i(j+1)}|_3 \oplus a_{i(j+1)}|_5 \oplus a_{i(j+1)}|_6) \\
&\quad \oplus (a_{i(j+2)}|_2 \oplus a_{i(j+2)}|_7) \\
&\quad \oplus (a_{i(j+3)}|_1 \oplus a_{i(j+3)}|_3 \oplus a_{i(j+3)}|_6 \oplus a_{i(j+3)}|_7) \\
&\quad \oplus (a_{i(j+4)}|_0 \oplus a_{i(j+4)}|_5 \oplus a_{i(j+4)}|_6) \oplus a_{i(j+5)}|_3 \\
&\quad \oplus (a_{i(j+6)}|_1 \oplus a_{i(j+6)}|_6 \oplus a_{i(j+6)}|_7) \oplus a_{i(j+7)}|_3
\end{aligned}
$$

$$
\begin{aligned}
b_{ij}|_4 = {} & a_{ij}|_4 \oplus (a_{i(j+1)}|_1 \oplus a_{i(j+1)}|_4 \oplus a_{i(j+1)}|_5 \oplus a_{i(j+1)}|_6 \oplus a_{i(j+1)}|_7) \\
& \oplus (a_{i(j+2)}|_3 \oplus a_{i(j+2)}|_7) \\
& \oplus (a_{i(j+3)}|_2 \oplus a_{i(j+3)}|_4 \oplus a_{i(j+3)}|_6 \oplus a_{i(j+3)}|_7) \\
& \oplus (a_{i(j+4)}|_1 \oplus a_{i(j+4)}|_5 \oplus a_{i(j+4)}|_6 \oplus a_{i(j+4)}|_7) \oplus a_{i(j+5)}|_4 \\
& \oplus (a_{i(j+6)}|_2 \oplus a_{i(j+6)}|_6 \oplus a_{i(j+6)}|_7) \oplus a_{i(j+7)}|_4 \\
b_{ij}|_5 = {} & a_{ij}|_5 \oplus (a_{i(j+1)}|_2 \oplus a_{i(j+1)}|_5 \oplus a_{i(j+1)}|_6 \oplus a_{i(j+1)}|_7) \oplus a_{i(j+2)}|_4 \\
& \oplus (a_{i(j+3)}|_3 \oplus a_{i(j+3)}|_5 \oplus a_{i(j+3)}|_7) \\
& \oplus (a_{i(j+4)}|_2 \oplus a_{i(j+4)}|_6 \oplus a_{i(j+4)}|_7) \oplus a_{i(j+5)}|_5 \\
& \oplus (a_{i(j+6)}|_3 \oplus a_{i(j+6)}|_7) \oplus a_{i(j+7)}|_5 \\
b_{ij}|_6 = {} & a_{ij}|_6 \oplus (a_{i(j+1)}|_3 \oplus a_{i(j+1)}|_6 \oplus a_{i(j+1)}|_7) \oplus a_{i(j+2)}|_5 \\
& \oplus (a_{i(j+3)}|_4 \oplus a_{i(j+3)}|_6) \oplus (a_{i(j+4)}|_3 \oplus a_{i(j+4)}|_7) \\
& \oplus a_{i(j+5)}|_6 \oplus a_{i(j+6)}|_4 \oplus a_{i(j+7)}|_6 \\
b_{ij}|_7 = {} & a_{ij}|_7 \oplus (a_{i(j+1)}|_4 \oplus a_{i(j+1)}|_7) \oplus a_{i(j+2)}|_6 \oplus (a_{i(j+3)}|_5 \oplus a_{i(j+3)}|_7) \\
& \oplus a_{i(j+4)}|_4 \oplus a_{i(j+5)}|_7 \oplus a_{i(j+6)}|_5 \oplus a_{i(j+7)}|_7
\end{aligned}
$$

A.5 64-Bit Vectors of the Result

These formulas show how to compute all eight 64-bit vectors of the result.

$$
\begin{aligned}
bs_0 = {} & as_0 \oplus rl^8(as_0) \oplus rl^8(as_5) \oplus rl^{16}(as_7) \oplus rl^{24}(as_0) \oplus rl^{24}(as_6) \\
& \oplus rl^{32}(as_5) \oplus rl^{40}(as_0) \oplus rl^{48}(as_6) \oplus rl^{56}(as_0) \\
bs_1 = {} & as_1 \oplus rl^8(as_1) \oplus rl^8(as_6) \oplus rl^{16}(as_0) \oplus rl^{24}(as_1) \oplus rl^{24}(as_7) \\
& \oplus rl^{32}(as_6) \oplus rl^{40}(as_1) \oplus rl^{48}(as_7) \oplus rl^{56}(as_1) \\
bs_2 = {} & as_2 \oplus rl^8(as_2) \oplus rl^8(as_5) \oplus rl^8(as_7) \oplus rl^{16}(as_1) \oplus rl^{16}(as_7) \\
& \oplus rl^{24}(as_0) \oplus rl^{24}(as_2) \oplus rl^{24}(as_6) \oplus rl^{32}(as_5) \oplus rl^{32}(as_7) \\
& \oplus rl^{40}(as_2) \oplus rl^{48}(as_0) \oplus rl^{48}(as_6) \oplus rl^{56}(as_2) \\
bs_3 = {} & as_3 \oplus rl^8(as_0) \oplus rl^8(as_3) \oplus rl^8(as_5) \oplus rl^8(as_6) \oplus rl^{16}(as_2) \\
& \oplus rl^{16}(as_7) \oplus rl^{24}(as_1) \oplus rl^{24}(as_3) \oplus rl^{24}(as_6) \oplus rl^{24}(as_7) \\
& \oplus rl^{32}(as_0) \oplus rl^{32}(as_5) \oplus rl^{32}(as_6) \oplus rl^{40}(as_3) \\
& \oplus rl^{48}(as_1) \oplus rl^{48}(as_6) \oplus rl^{48}(as_7) \oplus rl^{56}(as_3) \\
bs_4 = {} & as_4 \oplus rl^8(as_1) \oplus rl^8(as_4) \oplus rl^8(as_5) \oplus rl^8(as_6) \oplus rl^8(as_7) \\
& \oplus rl^{16}(as_3) \oplus rl^{16}(as_7) \oplus rl^{24}(as_2) \oplus rl^{24}(as_4) \oplus rl^{24}(as_6) \\
& \oplus rl^{24}(as_7) \oplus rl^{32}(as_1) \oplus rl^{32}(as_5) \oplus rl^{32}(as_6) \oplus rl^{32}(as_7) \\
& \oplus rl^{40}(as_4) \oplus rl^{48}(as_2) \oplus rl^{48}y(as_6) \oplus rl^{48}(as_7) \oplus rl^{56}(as_4) \\
bs_5 = {} & as_5 \oplus rl^8(as_2) \oplus rl^8(as_5) \oplus rl^8(as_6) \oplus rl^8(as_7) \oplus rl^{16}(as_4) \\
& \oplus rl^{24}(as_3) \oplus rl^{24}(as_5) \oplus rl^{24}(as_7) \oplus rl^{32}(as_2) \oplus rl^{32}(as_6) \\
& \oplus rl^{32}(as_7) \oplus rl^{40}(as_5) \oplus rl^{48}(as_3) \oplus rl^{48}(as_7) \oplus rl^{56}(as_5)
\end{aligned}
$$

$$
\begin{aligned}
bs_6 = {} & as_6 \oplus rl^8(as_3) \oplus rl^8(as_6) \oplus rl^8(as_7) \oplus rl^{16}(as_5) \oplus rl^{24}(as_4) \\
& \oplus rl^{24}(as_6) \oplus rl^{32}(as_3) \oplus rl^{32}(as_7) \oplus rl^{40}(as_6) \oplus rl^{48}(as_4) \\
& \oplus rl^{56}(as_6) \\
bs_7 = {} & as_7 \oplus rl^8(as_4) \oplus rl^8(as_7) \oplus rl^{16}(as_6) \oplus rl^{24}(as_5) \oplus rl^{24}(as_7) \\
& \oplus rl^{32}(as_4) \oplus rl^{40}(as_7) \oplus rl^{48}(as_5) \oplus rl^{56}(as_7)
\end{aligned}
$$

Author Index