Lecture Notes in Computer Science 2513

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

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

Robert Deng   Sihan Qing
Feng Bao   Jianying Zhou (Eds.)

Information and Communications Security

4th International Conference, ICICS 2002
Singapore, December 9-12, 2002
Proceedings

Springer

Volume Editors

Robert Deng
Feng Bao
Jianying Zhou
Labs for Information Technology
21 Heng Mui Keng Terrace, Singapore 119613
E-mail: {deng/baofeng/jyzhou}@lit.a-star.edu.sg

Sihan Qing
Engineering Research Center for Information Security Technology
Chinese Academy of Sciences
P.O. Box 8718, Beijing 100080, China
E-mail: qsihan@yahoo.com

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

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

ISSN 0302-9743
ISBN 3-540-00164-6 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2002
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 10870944 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 4th International Conference on Information and Communications Security (ICICS 2002). The three previous conferences were held in Beijing (ICICS 97), Sydney (ICICS 99) and Xian (ICICS 01), where we had an enthusiastic and well-attended event. ICICS 2002 is sponsored and organized by the Laboratories for Information Technology, Singapore, in co-operation with the Engineering Research Center for Information Security Technology of the Chinese Academy of Sciences and the International Communications and Information Security Association (ICISA).

During the past five years the conference has placed equal emphasis on the theoretical and practical aspects of information and communications security and has established itself as a forum at which academic and industrial people meet and discuss emerging security challenges and solutions. We hope to uphold this tradition by offering you yet another successful meeting with a rich and interesting program.

The response to the Call For Papers was overwhelming, 161 paper submissions were received. Therefore, the paper selection process was very competitive and difficult – only 41 papers were accepted and many good papers had to be rejected. The success of the conference depends on the quality of the program. We are indebted to our program committee members and the external referees for the wonderful job they did.

Before letting you enjoy the technical program, we would like to thank several individuals who skilfully and tirelessly put this conference together. First, Robert Deng would like to thank Prof. Sihan Qing for proposing to have this conference in Singapore. Next, our special thanks to the conference organizing committee, in particular Feng Bao (Local Arrangements), Jianying Zhou (Publicity and Publication), and Patricia Loh (Logistics and Registration). Without the hard work by all these folks, this conference would not have been possible.

Finally, we would like to thank the authors who submitted papers and the participants from all over the world who chose to honor us with their attendance.

September 2002

Robert Deng
Sihan Qing

ICICS 2002

Fourth International Conference on Information and Communications Security

Singapore December 9-12, 2002

Sponsored and organized by

Laboratories for Information Technology
(Singapore)

In co-operation with

Engineering Research Center for Information Security Technology
(Chinese Academy of Sciences)
and
International Communications and Information Security Association

Jean-Jacques Quisquater	UCL, Belgium
Mike Reiter	Carnegie Mellon University, USA
Bimal Roy	Indian Statistical Institute, India
Avi Rubin	AT&T Labs, USA
Ravi Sandhu	SingleSignOn.Net and George Mason University, USA
Shiuhpyng Shieh	National Chiao Tung University, Taiwan
Paul Syverson	Naval Research Lab, USA
Doug Tygar	University of California at Berkeley, USA
Vijay Varadharajan	Macquarie, Australia
Yumin Wang	Xidian University, China
Yiqun Lisa Yin	NTT MCL, USA
Yuliang Zheng	University of North Carolina at Charlotte, USA
Jianying Zhou	Laboratories for Information Technology, Singapore

Organizing Committee

Feng Bao	Laboratories for Information Technology, Singapore
Patricia Loh	Laboratories for Information Technology, Singapore
Jianying Zhou	Laboratories for Information Technology, Singapore

Table of Contents

Defenses against the Truncation of Computation Results of Free-Roaming Agents

Jeff S.L. Cheng and Victor K. Wei

Department of Information Engineering
The Chinese University of Hong Kong
{chengsl6,kwwei}@ie.cuhk.edu.hk

Abstract. Yee [Yee97] and Karjoth, *et al.* [KAG98] proposed schemes to protect the agent data. Various degrees of forward integrity were achieved by their results, among other benefits. A known vulnerability of their techniques is the truncation attack where two visited hosts (or one revisited host) can collude to discard the partial results collected between their respective visits. In this paper we propose several defenses against the truncation attack and the related growing-a-fake-stem ("stemming") attack for the protection of the partial computation results of free-roaming agents. In Protocol N1, we use a co-signing technique to prevent the two-colluder truncation attack. Generalizations of N1 can further raise the threshold number of colluding hosts we can defend. Protocol N2 does not prevent truncation or stemming. It detects stemming and identifies the exact pair of colluders for prosecution. Protocol N2 mainly relies on mutual authentication techniques.

Keywords: mobile agent security, malicious host, truncation attack, agent route protection, cryptographic protocols.

1 Introduction

Mobile agents are software programs that can autonomously migrate from host to host. During their execution, they work on behalf of their owners in order to fulfill their goals. *Free-roaming* mobile agents are free to choose their respective next hops dynamically based on the data they acquired from their past journeys.

Yee [Yee97] proposed using a Partial Result Authentication Code (PRAC) to ensure the integrity of the offers acquired from the hosts. In their scheme, an agent and its originator maintained a list of secret keys, or a key generating function. The agent used a key to encapsulate the collected offer and then destroyed the key. However, a malicious host may keep the key or the key generating function. When the agent revisits the host or visits another host conspiring with it, a previous offer or series of offers would be modified, without being detected by the originator.

Karjoth, *et al.* [KAG98] extended Yee's results. In the KAG scheme, each host generated a secret signature function for its successor and certified the corresponding verification predicate. Using the received signature/verification function pair, a host signed its partial result and certified a new verification function provided to the next host. Their scheme could resist the modification attack in Yee's scheme but not a *two-colluder truncation attack.* In this attack, a host with the agent at hand colludes with a previously visited host to discard all entries between the two visits. (Or a host revisited by the agent can discard all entries between its two visits.) The *grow-a-fake-*

stem ("stemming") attack is the simultaneous attack of the two-colluder truncation attack and the insertion of one or more chained fake offers in place of the truncated results.

Loureiro, *et al.* [LMP99] and Lam, *et al.* [LW02] proposed schemes to protect the agents. They noticed similar truncation issues in their protocol.

The *two-colluder truncation attack* and *stemming attack* have been identified in the literature as a known and open problem. To the best of our knowledge, there have been no effective solutions proposed to counter these attacks on a single agent. The main purpose of this paper is to propose several defenses against these attacks.

(During the preparation of our paper, Roth's results on the KAG protocols [Rot01a, Rot01b, Rot02] and the results of Vijil, *et al.* on identifying collusions [VI02] came to our attention. We plan to conduct further research on some of the issues raised there.)

Our Contribution. In this paper, we propose two new protocols (N1 and N2) for protecting the computation results of free roaming mobile agents:

- *Truncation defense*: We propose a new data collection protocol (N1) that prevents the two-colluder truncation attack in a free roaming agent. The main technique is to require an external party, typically the precedingly-visited host, to co-sign the agent migration. Therefore, two colluders are not sufficient to effect a truncation attack. At least three colluders are needed (not counting the revisitation case). Generalizations of our protocol can further raise the threshold number of colluding hosts needed to effect a truncation attack.
- *Stemming culprit identification*: Upon receiving the returning agent, the originator can detect the unsigned offers on the fake stem and discard them. In this paper, we are interested in enabling the originator to identify the exact pair of colluders who administered the stemming attack. We present Protocol N2 for accomplishing this. Our main technique is to require the sending host and the receiving host in an agent migration protocol to authenticate to each other. The authentication challenges-and-responses are encapsulated and carried along by the agent. By examining the validity (or the lack of) of the authentication data, the originator will be able to determine the identities of the two colluders and seek prosecution. We also propose using Protocol N2 to implement a spot-check scheme to defend the truncation attacks.

Both protocols N1 and N2 are motivated by, and are extensions of, protocols in [KAG98]. The techniques demonstrated in these protocols can be generally applied to other protocols.

Structure of the Paper. The rest of the paper is organized as follows: Section 2 specifies the notations and security properties in the agent environment. Section 3 reviews the previous work of Karjoth, *et al.* (K1) and studies the attacks against it. Section 4 gives the counter measurements to two-colluder truncation attacks (N1). Section 5 proposes a protocol (N2) to identify the culprits in stemming attacks. We have a concluding remark at section 6.

2 Notations and Assumptions

In this paper, we follow the comparison-shopping scenario studied in [KAG98]. An agent receiving a request from its originator S_0 will obtain a list of hosts, $\{S_1,\ldots,S_n\}$

from a directory server. This list may be updated during the agent's journey on the network. The agent will collect intermediate results o_i from a host S_i. This offer o_i is cryptographically encapsulated into O_i.

Table 1. Model Notations

$S_0 = S_{n+1}$	The originator.
S_i, $1 \le i \le n$	The hosts.
o_i, $1 \le i \le n$	The offers from S_i. The identities of S_i are explicitly specified in o_i.
O_i, $1 \le i \le n$	The encapsulated offers (cryptographically protected o_i) from S_i.
h_i, $1 \le i \le n$	Integrity check values associated with O_i.
V	A victim host whose identity is specified in a fake offer by the attackers.
$O_0, O_1,\ldots,O_n$	The chain of encapsulated offers from hosts $S_1, S_2,\ldots,S_n$.

Table 2. Cryptographic Notations

r_i	A random number generated by S_i.
$(\overline{\nu}_i, \nu_i)$	$\overline{\nu}_i$ is the private key of S_i and ν_i is the associated public key.
$(\overline{\mu}_i, \mu_i)$	The secret private/public key pair used by S_i.
$Enc_{\nu_0}(m, r_i)$	A message m asymmetrically encrypted with the public key ν_0 of S_0.
$Sig_{\overline{\nu}_i}(m)$	The signature of S_i on a message m using its private key $\overline{\nu}_i$.
$Ver(\sigma, \nu)$	A signature verification function with data recovery for signature σ and public key ν.
$\mathcal{H}(m)$	A one-way, collision-resistant hash function.
$m_1 \parallel m_2$	The concatenation of messages m_1 and m_2.
$[m]$	A message m sent via a confidential channel.
$Alice \rightarrow Bob$: m	Alice sending message m to Bob.

A public key infrastructure is assumed in the mobile agent environment. Each host S_i has a certified private/public key pair $(\overline{\nu}_i, \nu_i)$. Given a signature expressed as $Sig_{\overline{\nu}_i}(m)$, we assume that anyone could deduce the identity of S_i from it. The chain of encapsulated offers $O_1, O_2,\ldots,O_n$ is an ordered sequence. Each entry of the chain depends on some of the previous and/or succeeding members. A *chaining relation* specifies the dependency. The model notations are summarized in table 1 and the cryptographic notations are given in table 2.

Definition 1 An agent is defined as $\mathcal{A} = (I, C, \mathcal{S})$ where I is the identity, C is the code and S is the state of the agent. Both I and C are assumed to be static while $\mathcal{S}$ is variable. I is in the form of $(ID_{\mathcal{A}} \parallel Seq_{\mathcal{A}})$, where $ID_{\mathcal{A}}$ is a fixed identity bit string of the agent and $Seq_{\mathcal{A}}$ is a sequence number which is unique for each agent execution. The originator signs $Sig_{\overline{\nu}_0}(h_A)$, where $h_{\mathcal{A}} = H(I \parallel C)$ is the *agent integrity checksum* and $Sig_{\overline{\nu}_0}(h_A)$ is the *certified agent integrity checksum.* The agent carries this certified checksum, allowing the public to verify the integrity of I and C and deduce the identity of S_0.

Security Properties. We incorporate and extend the set of security properties defined in [KAG98]. Assume that an agent after visiting m undetermined hosts, $m \le n$, is captured by an attacker. This attacker, possibly the host S_{m+1}, obtains a chain of encapsulated offers $O_0, O_1, \ldots, O_m$. Some hosts excluding S_m may collude with the attacker. Letting $1 \le i \le m$ and assuming an agent environment with a security threshold parameter L, we have the following security properties.

S1. *Data Confidentiality*: Only the originator can extract the offers o_i from the encapsulated offers O_i.
S2. *Non-repudiability*: S_i cannot deny submitting o_i once S_0 receives o_i.
S3. *Forward Privacy*: No one except the originator can extract the identity information of the hosts S_i by examining the chain of encapsulated offers.
S4. *Strong Forward Integrity*: None of the encapsulated offers O_j, $j < m$, can be modified.
S5. *Publicly Verifiable Forward Integrity*: Anyone can verify the offer o_i by checking whether the chain is valid at O_i.
S6. *Insertion Defense*: No offer can be inserted at i unless explicitly allowed, i.e., S_{m+1}. It is not possible for S_{m+1} to insert more than one offers unless S_{m+1} colludes with some specific L hosts.
S7. *Truncation Defense*: Truncation at i is not possible unless some specific L hosts collude with S_i to carry out the attack.

Compared to the set of security properties defined in [KAG98], our notions strengthened the definitions of truncation resilience and insertion resilience to S6 and S7 such that it requires more than L colluders to carry out the attacks.

3 The Karjoth, *et al.* Protocol (K1)

We have combined the KAG protocols P2 and P4 to K1. The main technique in K1 is to set up a signature chain. To start the protocol, the originator S_0 randomly generates r_0 and a private/public key pair $(\bar{\mu}_1, \mu_1)$. S_0 also calculates a hashed value h_0 from r_0 and the identity of S_1. Then, S_0 encrypts a dummy offer using its own public key v_0. S_0 includes this encrypted value, h_0 and μ_1 in a dummy encapsulated offer O_0. Finally, S_0 signs O_0 by its private key $\bar{v}_0$ and sends O_0 and $\bar{\mu}_1$ to S_1.

When the agent arrives at host S_i, it contains the set of previously collected encapsulated offers O_k, $1 \le k < I$, and a secret key $[\bar{\mu}_i]$. From O_{i-1}, h_i can be calculated. Host S_i generates a private/public key pair $(\bar{\mu}_{i+1}, \mu_{i+1})$ and includes the public key μ_{i+1}, h_i and a signed-then-encrypted offer of o_i in the encapsulated offer O_i. The encapsulated offer O_i is further signed by $\bar{\mu}_i$.

Algorithmic description (K1)

- ***Agent creation protocol.*** S_0 should do the following:
  1. *Offer encapsulation and chaining relation*
     1.1. $h_0 = \mathcal{H}(r_0 \,||\, S_1)$
     1.2. $O_0 = Sig_{\bar{v}_0}(Enc_{v_0}(o_0, r_0), h_0, \mu_1)$

  2. *Agent transmission*
     2.1. Host $S_0 \rightarrow$ Host S_1: Π, O_0, [$\overline{\mu}_1$]
- ***Agent migration protocol*** *at* S_i, $1 \leq i \leq n$. S_i should do the following:
  3. *Agent verification*
     3.1. Receive $O_0, O_1, \ldots, O_{i-1}$, [$\overline{\mu}_i$]
     3.2. $Ver(O_0, v_0)$, recover h_0, μ_1
     3.3. $Ver(O_k, \mu_k)$, recover h_k, μ_{k+1} recursively, for $1 \leq k \leq i-1$
  4. *Offer encapsulation and chaining relation*
     4.1. $h_i = \mathcal{H}(O_{i-1} \parallel r_i \parallel S_{i+1})$
     4.2. $O_i = Sig_{\overline{\mu}_i}(Enc_{v_0}(Sig_{\overline{v}_i}(o_i), r_i), h_i, \mu_{i+1})$
  5. *Agent transmission*
     5.1. Host $S_i \rightarrow$ Host S_{i+1}: Π, $\{O_k \mid 0 \leq k \leq i\}$, [$\overline{\mu}_{i+1}$]

Remark 1: In the rest of this paper, we assume that the identity I, code C and $Sig_{\overline{v}_0}(h_A)$ of the agent are sent from S_i to S_{i+1}, but for simplicity, they are not specified in the algorithmic description.

Remark 2: We address the following issues that are not explicitly specified in the original KAG protocol:
- The self-generating private/public key pair ($\overline{\mu}_i, \mu_i$) is not reused.
- Each host has to verify $h_{\mathcal{A}}$, $Sig_{\overline{v}_0}(h_A)$ and O_0 in order to ensure three security issues. (1) The identity I and code C were not altered by any malicious hosts. (2) The host identity deduced from $Sig_{\overline{v}_0}(h_A)$ matches with that deduced from O_0. (3) I is not encrypted in O_0 such that each host checks that O_0 and $h_{\mathcal{A}}$ belong to the same execution of the same agent, i.e., no one can copy O_0 to other agents or another executions of the same agent. We omit this description in the above algorithmic description for simplicity.

  Both remarks 1 and 2 apply to protocols N1 and N2 in the following sections.

Two-colluder truncation attack. An attacker E captures an agent with encapsulated offers $O_0, O_1, \ldots, O_i, O_{i+1}, \ldots, O_m$ and colludes with a host S_i, who is visited by the agent before, such that S_i truncates all the offers after i. The attacker inserts an offer during the attack yielding the encapsulated offers $O_0, O_1, \ldots, O_i, O_E$. Note that S_i can rebuild the chaining relation using the new identity E in h_i, and sign the modified offer O_i using $\overline{\mu}_i$.

Growing a fake stem attack. After a truncation attack, S_i and S_{i+1} become two adjacent colluding hosts. Assume that they insert an offer in the name of a victim host V to the itinerary of an agent. In this case, S_i first inserts an offer O_i using its own identity in o_i and states V as its successor when computing h_i. The offer O_i is signed by $\overline{\mu}_i$. Then S_i generates a private/public key pair ($\overline{\mu}_V, \mu_V$) and inserts a fake offer O_V specifying the name of V in o_V. To create O_V, S_i first generates a new private/public key pair ($\bar{\mu}_{i+1}, \mu_{i+1}$), specifies S_{i+1} as the successor when computing h_V and then signs the fake offer O_V using $\bar{\mu}_V$. The following equations are obtained,

1. *Encapsulated Offer:*
   1.1. $O_i = Sig_{\bar{\mu}_i}(Enc_{v_0}(Sig_{\bar{v}_i}(o_i), r_i), h_i, \mu_V)$
   1.2. $O_V = Sig_{\bar{\mu}_V}(Enc_{v_0}(o_V, r_i), h_V, \mu_{i+1})$
   1.3. $O_{i+1} = Sig_{\bar{\mu}_{i+1}}(Enc_{v_0}(Sig_{\bar{v}_{i+1}}(o_{i+1}), r_{i+1}), h_{i+1}, \mu_{i+2})$
2. *Chaining Relation*
   2.1. $h_i = \mathcal{H}(O_{i-1} \| r_i \| V)$, (instead of $h_i = \mathcal{H}(O_{i-1} \| r_i \| S_{i+1})$)
   2.2. $h_V = \mathcal{H}(O_i \| r_V \| S_{i+1})$
   2.3. $h_{i+1} = \mathcal{H}(O_V \| r_{i+1} \| S_{i+2})$

After the agent returns to its originator, the originator would find out that o_V was not signed by V. However, it turns out that there could be two possibilities: (1) the host V intentionally inserted an offer O_V without signing on it, and (2) the host S_i inserts a fake offer in the name of V. The originator cannot accuse anyone because of the above uncertainties.

4 Truncation Defenses Based on Co-signing (N1)

To counter the problem of the two-colluder truncation attack, a *co-signing* mechanism is introduced. A host S_i needs a third party, typically its preceding host, to co-sign its encapsulated offer O_i. We will use this mechanism on K1 to demonstrate how it prevents the two-colluder truncation attack.

To implement the co-signing mechanism in N1, the encapsulated offer O_i is signed by S_{i-1}, instead of S_i. S_i has to send O_i to S_{i-1}. Host S_{i-1} signs O_i by the secret private key μ_{i-1} and then sends O_i back to S_i. In order to prevent a host from inserting two offers in a self-looping mode, each host S_i has to sign a signature σ_i on h_i using its own private key $\bar{v}_i$. When sending the agent to S_{i+1}, S_i sends all the encapsulated offers O_k, $0 \le k \le i$, $\bar{\mu}_{i+1}$, σ_{i-1} (received from S_{i-1}) and σ_i to S_{i+1}. The function of sending σ_{i-1} and σ_i to S_{i+1} is to allow S_{i+1} to verify that S_{i-1} and S_i are two distinct hosts. S_{i+1} attaches σ_{i-1} and σ_i in its encapsulated offer O_{i+1}, proving itself for completing this verification process.

Algorithmic description (N1)

- ***Agent creation protocol:***

1. *Offer encapsulation and chaining relation*
   1.1. $h_0 = \mathcal{H}(r_0 \| S_1)$
   1.2. $O_0 = Sig_{\bar{v}_0}(Enc_{v_0}(o_0, r_0), I, h_0, \mu_1)$
   1.3. $\sigma_0 = Sig_{\bar{v}_0}(h_0)$
2. *Agent transmission*
   2.1. Host $S_0 \rightarrow$ Host S_1: O_0, [$\bar{\mu}_1$, σ_0]

- ***Agent migration at S_1*** (all computations are performed at S_1 unless otherwise specified)

3. Agent verification

3.1. Receive O_0, $[\overline{\mu}_1, \sigma_0]$

3.2. $Ver(O_0, v_0)$, recover I, h_0, μ_1

3.3. $Ver(\sigma_0, v_0) \overset{?}{=} \text{correct}$

4. Interactive offer encapsulation and chaining relation

4.1. $h_1 = \mathcal{H}(O_0 \| r_1 \| S_2)$

4.2. $S_1 \to S_0$: $\{Enc_{v_0}(Sig_{\overline{v}_1}(o_1 \| \sigma_0), r_1), h_1, \mu_2\}$

4.3. $O_1 = Sig_{\overline{v}_0}(Enc_{v_0}(Sig_{\overline{v}_1}(o_1 \| \sigma_0), r_1), h_1, \mu_2)$, executed by S_0

4.4. $S_0 \to S_1$: O_1

4.5. $Ver(O_1, v_0) \overset{?}{=} \text{correct}$

4.6. $\sigma_1 = Sig_{\overline{v}_1}(h_1)$

5. Agent transmission

5.1. Host $S_1 \to$ Host S_2:$\{O_k \mid 0 \le k \le 1\}$, $[\overline{\mu}_2, \sigma_0, \sigma_1]$

- ***Agent migration at S_i***, $2 \le i \le n$, (all computations are performed at S_i unless otherwise specified)

6. Agent verification

6.1. Receive $O_0, O_1, \ldots, O_{i-1}$, $[\overline{\mu}_i, \sigma_{i-2}, \sigma_{i-1}]$

6.2. $Ver(O_0, v_0)$, recover I, h_0, μ_1

6.3. $Ver(O_1, v_0)$, recover h_1, μ_2

6.4. $Ver(O_k, \mu_{k-1})$, recover h_k, μ_{k+1} recursively for $2 \le k \le i - 1$

6.5. $Ver(\sigma_{i-2}, v_{i-2}) \overset{?}{=} \text{correct}$

6.6. $Ver(\sigma_{i-1}, v_{i-1}) \overset{?}{=} \text{correct}$

6.7. $S_{i-2} \overset{?}{\neq} S_{i-1}$

7. Interactive offer encapsulation and chaining relation

7.1. $h_i = \mathcal{H}(O_{i-1} \| r_i \| S_{i+1})$

7.2. $S_i \to S_{i-1}$: $\{Enc_{v_0}(Sig_{\overline{v}_i}(o_i \| \sigma_{i-2} \| \sigma_{i-1}), r_i), h_i, \mu_{i+1}\}$

7.3. $O_i = Sig_{\overline{\mu}_{i-1}}(Enc_{v_0}(Sig_{\overline{v}_i}(o_i \| \sigma_{i-2} \| \sigma_{i-1}), r_i), h_i, \mu_{i+1})$, executed by S_{i-1}

7.4. $S_{i-1} \to S_i$: O_i

7.5. $Ver(O_i, \mu_{i-1}) \overset{?}{=} \text{correct}$

7.6. $\sigma_i = Sig_{\overline{v}_i}(h_i)$

8. Agent transmission

8.1. Host $S_i \to$ Host S_{i+1}:$\{O_k \mid 0 \le k \le i\}$, $[\overline{\mu}_{i+1}, \sigma_{i-1}, \sigma_i]$

If the any one of the above agent verifications fails, S_i rejects the agent. If S_i detects that S_{i-1} has sent the same agent twice, S_i rejects the agent.

Remark 1: the host $S_{i\text{-}1}$ will store the records of the departed agents in such a way that it would only sign once for S_i on the partial results of a particular departed agent.

Remark 2: at the end of shopping, the agent will eventually leave the last host S_n and return to the originator. The originator may either send a message to host S_n to confirm the return of the agent to release S_n from waiting for the hand shaking process or generate a dummy offer to complete the hand shaking process with S_n.

Remark 3: we assume that there is an authentication protocol between S_i and $S_{i\text{-}1}$ such that both hosts are clear about the identity of the agent in the co-signing process.

Protocol N1 can be easily generalized or modified in the following possible ways:

1. The agent carefully designs its migration path such that it travels to hosts of opposing camps alternatively, making the collusion between two adjacent hosts less likely.
2. Include L preceding hosts in the co-signing process to prevent the L-colluder truncation attack.
3. Have a trusted third party (TTP) group in the mobile agent environment such that an agent can request a neighborhood TTP to co-sign the offer. This completely prevents truncation attack.

Security Analysis. N1 achieves the following security properties:

– (*S1*) *Data Confidentiality*. Each host S_i encrypts its signed offer by the originator's public key in $Enc_{v_0}(Sig_{\bar{v}_i}(o_i), r_i)$ which can only be extracted by the originator.

– (*S2*) *Non-repudiability*. Each host S_i signs its offer o_i by its private key in $Sig_{\bar{v}_i}(o_i)$. S_i cannot deny its offer after the agent returns to the originator.

– (*S3*) *Forward Privacy*. The host identity S_i is encrypted using the originator's public key in $Enc_{v_0}(Sig_{\bar{v}_i}(o_i), r_i)$. Therefore, no one except the originator can extract S_i's identity from the agent itinerary. Also, a random number r_i is used in computing the checksum h_i, therefore examining h_i reveals no identity information.

– (*S4*) *Strong Forward Integrity*. Assume that the attacker leaves O_m intact but changes the (m-1)th offer $O_{m\text{-}1}$ to $O_{m\text{-}1}'$. Since $h_m = \mathcal{H}(O_{m\text{-}1} \| r_m \| S_{m+1})$ and h_m is signed in O_m, the value of h_m cannot be changed in order to keep a valid signature in O_m. Thus, it requires to have $\mathcal{H}(O_{m\text{-}1}' \| r_m \| S_{m+1}) = \mathcal{H}(O_{m\text{-}1} \| r_m \| S_{m+1})$. This violates our assumption that the hashing function $\mathcal{H}(\cdot)$ is collision resistant. Alternatively, the attacker has to change O_m in response to the change in $O_{m\text{-}1}$, violating our assumption that O_m is not changed. Similarly, in order to modify any offer O_j, $0 \le j < m$, the attacker has to modify O_{j+1}. By induction, strong forward integrity is achieved.

– (*S5*) *Publicly Verifiable Forward Integrity*. As shown in steps 6.2 ~ 6.5, anyone can verify the encapsulated offers.

– (*S6*) *Insertion defense*. Following similar reasoning as in the above analysis, anyone obtaining a chain of encapsulated offers O_0, O_1,…,O_m cannot insert an offer at i, $0 \le i \le m$. This is because the chaining relation will not be respected after the insertion point. Moreover, S_{m+1} cannot insert more than one offer as S_m co-signs only one offer with respect to the particular agent from S_{m+1}.

– (*S7*) *Truncation defense*. Assume that S_i colludes with an attacker to carry out a truncation attack. There are two scenarios in which S_i may truncate the itinerary. In the first scenario, S_i truncates O_k, $i \le k \le m$. In the second scenario, S_i truncates O_k, $i < k \le m$. For the first scenario, S_i has to hand shake with S_{i-1} in order to have S_{i-1} to co-sign the new offer O_i'. This turns out to be impossible because S_{i-1} will detect that it is the second request for co-signing an offer to the same agent. In the second scenario, S_i does not change O_i, or the chaining relation will be altered. S_i must send the agent to S_{i+1} again in order to keep the chain relation $h_i = \mathcal{H}(O_{i-1} \,||\, r_i \,||\, S_{i+1})$. S_{i+1} will detect this agent resent action and thus reject the agent.

Drawbacks. N1 leaves residual duty for the host S_{i-1}, who has to keep records for the departed agents and act as a co-signer for S_i. It also sacrifices the privacy of S_{i-2} to S_i so as to allow S_i to verify that S_{i-1} and S_{i-2} are distinct (steps 6.5 ~ 6.7).

5 Culprit Identification Schemes in the Stemming Attack

We modify K1 by enforcing a *two-way authentication mechanism* in N2. With this mechanism, we ensure that the two hosts involved in an agent transmission protocol must mutually authenticate themselves, and the authentication data is carried by the agent such that the originator can accuse the culprits in a stemming attack.

When sending an agent from S_i to S_{i+1}, S_i must first obtain a *backward authentication signature* $\sigma_{B,i+1}$ from S_{i+1}, in which S_{i+1} signs ($S_{i+1} \,||\, h_i$) using its own private key $\overline{v}_{i+1}$. Then S_i sends a *forward authentication signature* $\sigma_{F,i}$ to S_{i+1}, where $\sigma_{F,i}$ is a signature of ($S_i \,||\, h_i$) signed by the secret key $\overline{v}_i$. Both hosts will verify the received authentication signatures using public keys v_i and v_{i+1} respectively. The authentication data $\sigma_{F,i-1}$ and $\sigma_{B,i+1}$ will be signed by S_i and encapsulate in O_i, where $\sigma_{F,i-1}$ is a forward authentication signature obtained from S_{i-1}. With the two-way authentication mechanism, S_i cannot deny being the preceding host of S_{i+1}. Similarly, S_{i+1} cannot deny being the successor of S_i.

Algorithmic description (N2)

- ***Agent creation protocol*** (all computations are done in S_0 unless otherwise specified)

1. Interactive offer encapsulation and chaining relation

1.1. $h_0 = \mathcal{H}(r_0 \,||\, S_1)$

1.2. $S_0 \to S_1$: h_0

1.3. $\sigma_{B,1} = Sig_{\overline{v}_1}(S_1 \,||\, h_0)$, executed in S_1

1.4. $S_1 \to S_0$: $[\sigma_{B,1}]$

1.5. $Ver(\sigma_{B,1}, v_1)$

1.6. $O_0 = Sig_{\overline{v}_0}(Enc_{v_0}(o_0, \sigma_{B,1}, r_0), I, h_0, \mu_1)$

1.7. $\sigma_{F,0} = Sig_{\overline{v}_0}(S_0 \,||\, h_0)$

2. Agent transmission

2.1. Host $S_0 \to$ Host S_1: O_0, $[\overline{\mu}_1, \sigma_{F,0}]$

- ***Agent migration protocol** at* S_i, $1 \le i < n$

3. *Agent verification*
   3.1. Receive $O_0, O_1, \ldots, O_{i-1}$, $[\bar{\mu}_i, \sigma_{F,i-1}]$
   3.2. $Ver(O_0, v_0)$, recover I, h_0, μ_1
   3.3. $Ver(O_k, \mu_k)$, recursively recover h_k, μ_{k+1}, for $1 \le k \le i - 1$
   3.4. $Ver(\sigma_{F,i-1}, v_{i-1}) \overset{?}{=} \text{correct}$, recover h_{i-1}
   3.5. h_{i-1} (recovered in step 3.3) $\overset{?}{=} h_{i-1}$ (recovered in step 3.4) $\overset{?}{=} h_{i-1}$ (used in computing $\sigma_{B,i}$, e.g. in step 1.3 for $i = 1$)
4. *Interactive Offer encapsulation and chaining relation*
   4.1. $h_i = \mathcal{H}(O_{i-1} \parallel r_i \parallel S_{i+1})$
   4.2. $S_i \rightarrow S_{i+1}$: h_i
   4.3. $\sigma_{B,i+1} = Sig_{\bar{v}_{i+1}}(S_{i+1} \parallel h_i)$, executed in S_{i+1}
   4.4. $S_{i+1} \rightarrow S_i$: $[\sigma_{B,i+1}]$
   4.5. $Ver(\sigma_{B,i+1}, v_{i+1})$
   4.6. $\sigma_{F,i} = Sig_{\bar{v}_i}(S_i \parallel h_i)$
   4.7. $O_i = Sig_{\bar{\mu}_i}(Enc_{v_0}(Sig_{\bar{v}_i}(o_i \parallel \sigma_{F,i-1} \parallel \sigma_{B,i+1}), r_i), h_i, \mu_{i+1})$
5. *Agent transmission*
   5.1. Host $S_i \rightarrow$ Host S_{i+1}: $\{O_k \mid 0 \le k \le i\}$, $[\bar{\mu}_{i+1}, \sigma_{F,i}]$

If the any one of the above agent verification steps fails, S_i rejects the agent. Also, S_i will check if the same agent with the same chain of encapsulated offers has visited it before. If yes, S_i rejects the agent.

Stemming attack. Consider that case where either S_i or V inserts an unsigned offer in a chain of encapsulated offers $O_1, O_2, \ldots, O_i, O_V, O_{i+1}$ with the following equations:

1. *Encapsulated offer:*
   1.1. $O_i = Sig_{\bar{\mu}_i}(Enc_{v_0}(Sig_{\bar{v}_i}(o_i \parallel \sigma_{F,i-1} \parallel \sigma_{B,V}), r_i), h_i, \mu_V)$
   1.2. $O_V = Sig_{\bar{\mu}_V}(Enc_{v_0}(o_V, r_V), h_V, \mu_{i+1})$
   1.3. $O_{i+1} = Sig_{\bar{\mu}_{i+1}}(Enc_{v_0}(Sig_{\bar{v}_{i+1}}(o_{i+1} \parallel \sigma_{F,V} \parallel \sigma_{B,i+2}), r_{i+1}), h_{i+1}, \mu_{i+2})$
2. *Chaining relation*
   2.1. $h_i = \mathcal{H}(O_{i-1} \parallel r_i \parallel V)$
   2.2. $h_V = \mathcal{H}(O_i \parallel r_V \parallel S_{i+1})$
   2.3. $h_{i+1} = \mathcal{H}(O_V \parallel r_{i+1} \parallel S_{i+2})$

Note that both $\sigma_{B,V}$ in O_i and $\sigma_{F,V}$ in O_{i+1} could be fraudulent because S_i cannot sign in the name of V.

Stemming detection. Upon the agent's return, its originator can extract the offers from the chain of encapsulated offers and detect an unsigned offer o_V with false authentication data in O_i, O_V and O_{i+1}, the originator can detect the stemming attack.

Culprit identification. We use $T(V, S_i)$ to denote the case when an authentication signature from V to S_i is genuine and $F(V, S_i)$ to denote the fraudulent case. Culprits are identified according to the following situations:

1. if $F(V, S_i)$ and $F(V, S_{i+1})$, then S_i and S_{i+1} will be accused for inserting a fake offer in the name of V.
2. if $T(V, S_i)$ and $T(V, S_{i+1})$, then V will be accused for not signing its offer o_V.
3. if $T(V, S_i)$ and $F(V, S_{i+1})$, then V will be accused for not signing its offer and S_{i+1} will be accused for not enforcing the verification protocol.
4. if $F(V, S_i)$ and $T(V, S_{i+1})$, then V will be accused for not signing its offer and S_i will be accused for not enforcing the verification protocol.

Remark: We assume that neither $\sigma_{B,V}$ nor $\sigma_{F,V}$ was signed by S_i or S_{i+1}, otherwise, they will be directly identified by verifying these signatures. Also, we assume that there is no genuine authentication information stored in O_V, otherwise such information will allow the originator to directly locate the culprits.

Truncation defense (spot check). We can detect a truncation attack by comparing a truncated agent with its itinerary before the truncation. In this scheme, an agent will visit a trusted third party known as the spot check center (SCC) intermittently. When an agent $\mathcal{A}$ visits the SCC for the first time, the SCC keeps a clone $\mathcal{A}_C$ of the agent and allows the original agent $\mathcal{A}$ to continue its journey. Upon the agent's re-visitation, the SCC compares the chain of encapsulated offers in $\mathcal{A}$ with that in $\mathcal{A}_C$. If the chain of encapsulated offers in $\mathcal{A}_C$ matches with the first portion of that of $\mathcal{A}$, the SCC updates $\mathcal{A}_C = \mathcal{A}$ and allow $\mathcal{A}$ to continue the next host. Otherwise, the SCC immediately sends both $\mathcal{A}$ and $\mathcal{A}_C$ back to their originator for truncation detection. When the originator receives two agents $\mathcal{A}$ and $\mathcal{A}_C$ from the SCC, the originator can use the above culprit identification techniques to figure out if there are culprits in each itinerary. The originator then compares the two itineraries and identifies the last common host in the two itineraries as the culprit for the truncation attack.

Drawbacks. The integration of the mutual authentication protocol leads to an interactive offer encapsulating protocol in N2. However, if the agent system has an existing mutual authentication scheme, the integration simply swaps the sequence of the offer encapsulation protocol and the authentication protocol.

6 Conclusion

In this paper, we proposed several defenses against the truncation attacks and grow-a-fake-stem attacks for the purpose of better protection of the computation results of free-roaming agents. Protocol N1 prevents the two-colluder truncation attack. It can also be generalized to prevent the L-colluder truncation attack. Protocol N2 detects the grow-a-fake stem attack and identifies the colluding pair for prosecution. The main technique of Protocol N1 is co-signing, and that of Protocol N2 is two-way authentication. These techniques can be generally applied to other protocols.

References

[Yee97] B.S. Yee, "A Sanctuary for Mobile Agents," *Technical Report CS97-537*, UC San Diego, Department of Computer Science and Engineering, April 1997.

[KAG98] G. Karjoth, N. Asokan, and C. Gülcü, "Protecting the computation results of free-roaming agents," *Proc. Second International Workshop on Mobile Agents (MA '98), K. Rothermel and F. Hohl, editors, LNCS 1477*, pp. 195 - 207, Springer-Verlag, 1998.

[LMP99] S. Loureiro, R. Molva, and A. Pannetrat, “Secure data collection with updates,” In Proceedings of the *Workshop on Agents on Electronic Commerce - First Asia Pacific Conference on Inteligent Agent technology*, Hong-Kong, December 1999.

[Rot01a] V. Roth, "Programming Satan's agents," in *Proc 1st International Workshop on Secure Mobile Multi-Agent Systems, (Montreal, Canada)*, May 2001.
<URL http: //www.dfki.de/kuf/semas/semas-2001/>

[Rot01b] V. Roth. “On the robustness of some cryptographic protocols for mobile agent protection,” In *Proc. Mobile Agents 2001, LNCS 2240*. Springer Verlag, December 2001.

[Rot02] V. Roth. *Empowering Mobile Software Agents.* Submitted to 6th IEEE Conference on Mobile Agents, Barcelona, Spain, 2002.

[LW02] T.C. Lam, and V.K. Wei, “A Mobile Agent Clone Detection System with Itinerary Privacy,” *11th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE ’02)* 2002.

[VI02] E. Vijil, S. Iyer, “Identifying Collusions: Co-operating Malicious Hosts in Mobile Agent Itineraries,” in *Second International Workshop on Security of Mobile Multiagent Systems, SEMAS ’02*, Italy, July 2002.

A Distributed Dynamic μFirewall Architecture with Mobile Agents and KeyNote Trust Management System

Hai Jin, Feng Xian, Zongfen Han, and Shengli Li

Internet and Cluster Computing Center,
Huazhong University of Science and Technology, Wuhan, 430074, China
hjin@hust.edu.cn

Abstract. Due to end-to-end design principle in distributed applications, many emerging security problems could not be solved by conventional security technologies, such as firewalls and IDSs. To address these problems, we present a distributed dynamic μFirewall architecture based on mobile agents and KeyNote trust management system. In this architecture, KeyNote trust management system provides the scalable distributed control capability and supports a mechanism called "policy-updates on demand". Mobile agents implement dynamic security policy reconfiguration and enhance the scalability. Each μFirewall is built with a packet filter and DTE-enhanced evaluator to enforce policy at the end points. A distributed intrusion detection and response (DIDR) system supports dynamic security capabilities and provides fast response to attacks from all possible sources. Our architecture is scalable, topology independent, and intrusion-tolerant.

1 Introduction

With the prospect of end-to-end computing, such as IPSec, SOAP, properties that hold end-to-end must be provided by applications at the endpoints. STRONGMAN [12] at the University of Pennsylvania is the first architecture for providing strong security services which satisfies end-to-end property. It adopts the trust-management authorization model, policy compliance checker local to the enforcement points, a simple model for providing policy composition, and the concept of lazy binding of policies.

However, the core foundation of the pervasive security technologies (firewalls, IDSs) for distributed applications is arguably the end-to-end design principle. Firewalls enforce a single security policy at network boundaries, which has led to many problems such as performance degradation, preventing inside attacks and failing to handle dynamic communication (FTP, RealAudio). IDSs also face some limits such as difficultly screening end-to-end encrypted traffic.

It is clear that the placement of these security functionalities should be moved deeper directly into the endpoints. Distributed firewall [4][11] proposed by Steven Bellovin et.al. is the first attemp to meet this need. In their implementation, the policy is distributed to all endpoints where it is authenticated and then enforced, thus making security an end-to-end property.

In this paper, we tie together multiple security components to build a solid end-to-end security frame. In our approach, we employ KeyNote trust management [5][6] to construct and interpret the trust relationships among the hierarchical security components. Mobile agents facilitate policy delivery without slowing down the traffic flow. Security policy is enforced by low-overhead µFirewalls at endpoints. Besides, Our architecture integrates a distributed intrusion detection and response (DIDR) system to provide more aggressive response to intrusions from all possible sources, either inside or outside.

The following part of the paper will delve further into the aforementioned components, and finally give a primitive evaluation of our architecture for its security, performance and scalability.

2 Policy Management Model

In our approach, the hierarchical management model is adopted shown in Figure 1. *Central Policy Administrator* (CPA) maintains global consistent policy. The central policy can be divided into multiple policy domains managed by *Sub-Policy Administrator* (SPA). The depth of the management tree is decided practically, such as department within an organization or topology unit (VLAN, LAN, etc).

As shown in Figure1, µFirewall enforces network security policy at each individual network endpoint (SPA, CPA, gateways, routers, *etc.*). *Intrusion Detection Agent* (IDA) and *Intrusion Response Agent* (IRA) compose an *distributed intrusion detection and response* (DIDR) system to repel attacks from everywhere. In our architecture, we employ KeyNote to express delegation of trust.

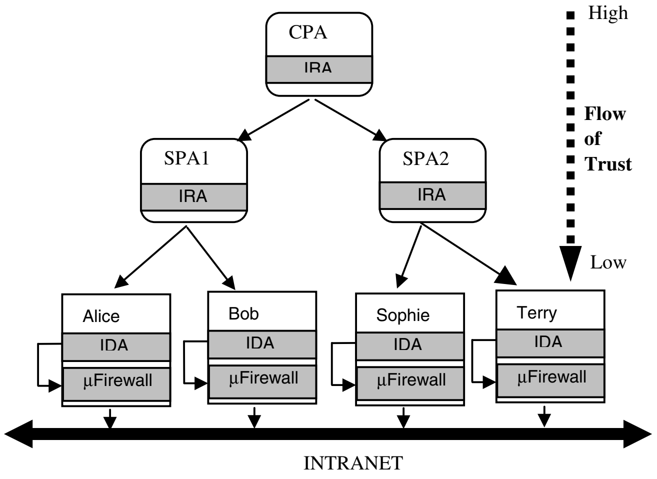


Fig. 1. Distributed Dynamic µFirewall Architecture

2.1 Delegation of Trust

Delegation of trust in our system is done to allow more than one SPA or CPA to make policy updates and to protect the system from single point of failures. Besides, it lets the hosts to identify its parents and trust only its parents for policy updates. Delegation of trust is done through issuing the public key certificates [8] and KeyNote credentials in a hierarchal manner.

The hierarchy of trust also defines how the system responds to intrusions. IRA in CPA prompts the system-level response. The μFirewalls come under the SPAs and take host-level reaction. This hierarchical response scheme will be illustrated in Section 5.

Figure 2 shows an example credential where CPA delegates authorization to the SPA1. This credential would allow the SPA1 to report any distributed intrusion events and request for a response under the system level. In Figure 3, SPA1 signs a credential indicating that Alice is authorized to report any probing events and consequential response is under the domain level.

```
KeyNote-Version: 2
Comment: SPA1 is trusted to request a system-level response.
Authorizer: "CPA-Key"
Licensees: "SPA1-Key"
Conditions: (app_domain == "ResponseIntrusion") &&
            (intrusion_type== “Distributed_Attack”)
            ( @response_level<= SYSTEM_LEVEL);
Signature: …<signed by CPA>…
```

Fig. 2. Credential Example of SPA1

```
KeyNote-Version: 2
Comment: Alice is trusted to request a domain-level response.
Authorizer: "SPA1-Key"
Licensees: "Alice-Key"
Conditions: (app_domain == "ResponseIntrusion") &&
            (intrusion_type== “Probe_Attack”)
            ( @response_level<= DOMAIN_LEVEL);
Signature: …<signed by SPA1>…
```

Fig. 3. Credential Example of Alice

2.2 Policy Cache and Update on Demand

In a centralized architecture, gateway firewall enforces a single security policy at network boundaries. To implement the policy globally, the network topology must be restricted to pass all traffic through the firewall. Consequently, gateway firewall is generally overloaded with processing requests and mediating access. In our approach, however, we distribute the global policy rules and form a multiple-level policy cache structure. The SPA is the policy cache of CPA, and each enforcement point is sub-level policy cache of the SPA. Each enforcement point or SPA keeps partial policy of the higher administrator. For example, web servers within the security perimeter only support web-related security policy rules. Since every packet is directly mediated and checked against the policy cache at endpoints, performance bottleneck brought out by gateway firewall is eliminated.

With the growth number of users and enforcement points, the global policy pertaining to both also needs scale up. Hence, we need update policy as cheap as possible. KeyNote trust-management system can satisfy our needs. In KeyNote, policy can be translated into credentials signed by SPA or CPA. Since policy is expressed in the form of credentials, policy need not be distributed synchronously to the enforcement points. Enforcement points evaluate credentials passed from remote users and build relevant rules for users to gain access to controlled resources. This property, namely *lazy policy instantiation*, enables enforcement points to update policy rules as needed. Moreover, IDAs and IRAs are also involved to update security policy once they find intrusion happened. They provide feedback to enforcements points and elevate the level of security policy. To sum up, the *lazy instantiation* and *dynamic security* mechanism make policy updates *on demand* and allow our system to scale well with the number of enforcement points.

3 μFirewall

In our prototype, the μFirewall is implemented at the kernel level using iptables and DTE package. These two security components together provide a *defense in depth* protection. Figure 4 shows our μFirewall module built at the kernel space.

iptables is a package to implement packet-filtering rules in a Linux kernel of version 2.4 or higher. iptables has a filter table and provides mechanisms to specify the packet-filtering rules. According to these rules, iptables can inspect all network traffic and drop those illegal packets.

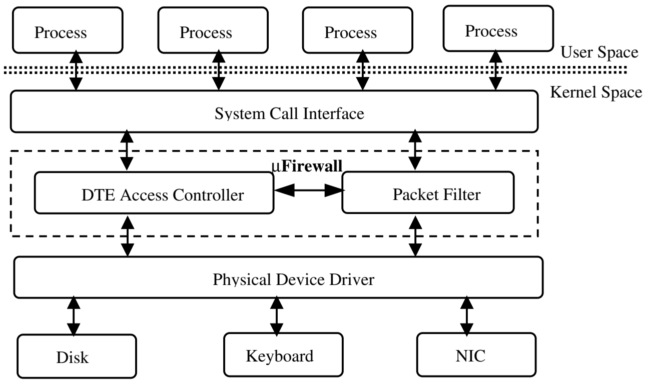


Fig. 4. μFirewall Module at Kernel Space

DTE [2] is an enhanced form of type enforcement technology, which is a table-oriented mandatory access control mechanism. DTE has many advantages: flexibility, simplicity, operation system interoperability, binary application compatibility. Using the DTE package, we can export some security-sensitive service (e.g., NFS, TFTP) safely available through security perimeter, which was impossible in gateway firewall. Also we can prevent iptables module from being tampered by *unwitting insiders* or malicious attackers.

4 Mobile Agents for Policy Distribution

To decrease client-server interactions between the policy administrator and the client hosts when a large number of hosts have to be updated with a new policy, we use *mobile agents* to facilitate the fast delivery without slowing down the traffic flow. Moreover, mobile agents have a lot of advantages, all related to cost-effectiveness, fast response time, and easy to make security policy changes.

As shown in Figure 5, every agent contains the following data: a *policy certificate*, an *itinerary*, an *interaction sequence*, and a small *state memory*. The policy certificate contains the policy update signed by CPA or SPA, which is expressed in the form of KeyNote credential. The policy certificate can also be used to authenticate agents and prevent any attacks on the host by malicious agents [7]. The itinerary is a list of hosts to visit in order to make policy updates. The host interaction sequence tells what the agent is supposed to do during its visits of successive hosts along the way. The state memory is reserved for the agent to store its state information, such as account of various interactions.

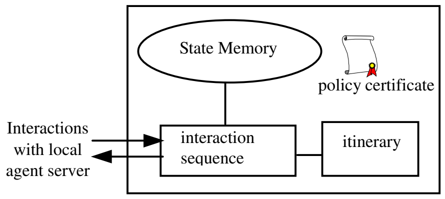


Fig. 5. Mobile Policy Agent

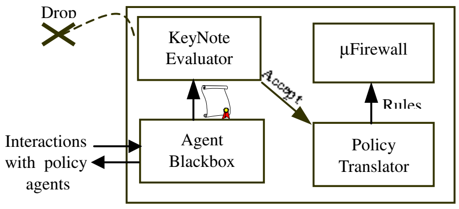


Fig. 6. Policy Update Process

When updates security policy, administrator creates some policy agents and dispatch them to all hosts in the same security domain. The policy agents carry the policy certificates and specify itinerary as the IP addresses of hosts to visit. After dispatched, these agents automatically update the policy on the µFirewalls visited. Once a policy agent reaching a host, as depicted in Figure 6, its policy certificate is verified by KeyNote evaluator. If the positive result is returned, the policy certificate will be passed to and decoded by a *policy translator*, which translates the policy into low-level policy language (packet-filtering rules). The rules are then executed on the local host to make the specified policy update to the kernel's packet filtering module.

Figure 7 lists a policy certificate expressed in KeyNote. In this certificate, SPA1 distributes new policy to its subordinates (i.e. Alice, Bob). The “version” attribute indicates the version number of the policy, which protects against a replay attack. The value of the “lifetime” attribute specifies the valid period of this certificate. The policy certificate extends the Simple Public Key Infrastructure (SPKI) [8] and can be considered as an *attribute certificate*. The PKI is used in an organization to bind the public keys to particular entities. This is called a *public key certificate* (PKC). A PKC can be considered to be like a passport: it identifies the holder, tends to last for a long time, and should not be trivial to obtain. An AC is more like an entry visa: it is typically issued by a different authority and does not last for a long time. Mobile agents carry this KeyNote-expressed attribute certificate and relevant PKC to prove their identities and the privileges.

```
KeyNote-Version: 2
Comment: SPA1 specifies a policy update to block traffic from 10.0.0.10
Authorizer: "SPA1-Key"
Licensees: "Alice-Key"|| "Bob-Key"
Conditions: (app_domain == "PolicyUpdate") &&
             (@version==20020320)&&    # version information for this certificate
             (@timestamp==190103)&&   # creation time of this certificate (19:01:03)
             (@lifetime<60)&&         # validity period of the certificate (<60 seconds)
             (policy== “block Src_IP(10.0.0.10)” );   # policy updates go here
Signature: …<signed by SPA1>…
```

Fig. 7. An Policy Certificate Example Carried by Agents from SPA1

The local host (i.e. Alice, Bob) has a policy as shown in Figure 8. The policy states that SPA1 will specify end-host security policy when the value given in the “version” attribute is larger than 20020320. On accepting a mobile agent dispatched from SPA1, the host (say, Alice) will check the validity of the policy certificate using the lifetime and version information on the certificate and also by verifying the digital signature of the certificate which is signed by SPA1.

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensees: "SPA1-KEY"
Conditions: (app_domain == "PolicyUpdate") &&(@version>=20020320)
          # version number for the next policy certificate is no less than 20020320
```

Fig. 8. Local Policy in Alice or Bob

5 DIDR Sysem

DIDR system includes two components: *intrusion detection agent* (IDA) and *intrusion response agent* (IRA). The former resides at every protected endpoints and supports detection and *host-level* response. The latter runs both at SPA to perform *domain-level* response and at CPA to take *system-level* response. The DIDR system provides a communication bridge between μFirewalls to response some distributed cooperative attacks.

Intrusion Detection Agent screens all network packets and local activities to detect attacks according to some intrusion-related *signatures* at end points. IDA is implemented using light-weight intrusion detection software such as *lids* and *portscan*. For those locally detected attacks, IDA takes a local response decision (e.g., kill the connection, install filtering rules, disable the user account) based on the attack type, attack certainty, attack severity, and local policy constraints. Although the IDA can specify its desired blocking action, the IRA may perform a different action. Hence, most of the local response implementations are for a short period of time (on the order of minutes), with the objective of providing the time needed for the IRA to develop a better long-term response.

Intrusion Response Agent collects alarm messages from all hosts within the same policy domain and facilitates security policy reconfiguration to reduce the risk of furthers penetrations. According to collected messages, IRA can then correlate them to gain a better overall picture of the situation, and also issue response directives back to individual nodes to either remove an unnecessary response, or update security policy of the whole domain. Once SPA detects a severe *cooperative intrusion* [1] attempt, it distributes an attack report to policy administrator at higher level (e.g., CPA) who can then help trace the attack path and take a system-level respond to the intrusion.

Figure 9 below illustrates how DIDR system accomplishes hierarchical intrusion response.

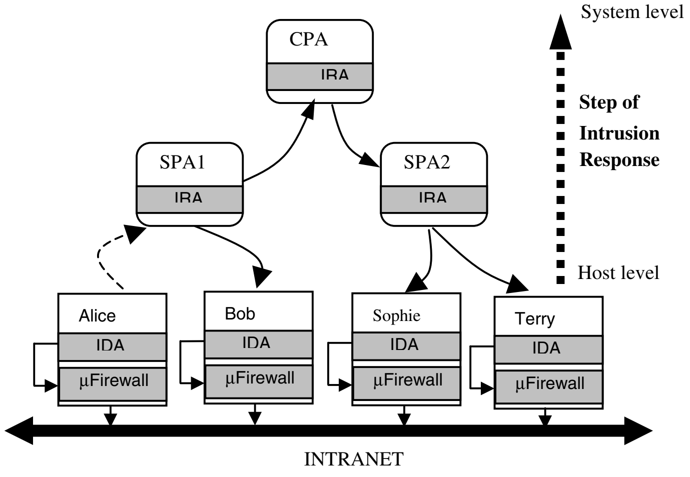


Fig. 9. Hierarchical Intrusion Response

First (Step1), an external or internal hacker penetrated an "internal" host Alice, and the attack is detected by IDA in Alice. IDA in Alice adds a temporarily blocking rules to μFirewall. Then, IDA in Alice passes its local credential, illustrated in Figure 3, to higher administrator SPA1 and reports the event.

Then (Step 2), SPA1 collectively determines a domain-level response and dispatches policy agents carrying the KeyNote-expressed attribute certificate, as shown in Figure 7, to update the security policy of its subordinate hosts (Alice, Bob). To some infectious attacks (e.g., viruses), SPA1 submits its credential shown in Figure 2 to CPA and informs CPA to take a broad range action.

Next (Step 3), CPA takes a system-level response and distributes the policy updates to all SPAs (SPA1, SPA2).

Last (Step 4), each SPA continues to distribute the policy updates to hosts as Step2.

6 Security, Performance, and Scalability Evaluation

Security, performance, scalability are three major considerations in evaluating security systems for Internet. In this section, we evaluate our distributed μFirewall (DMFW) and compare it to centralized firewall (CFW) in detail.

Our experimental environment are based on x86 architecture machines running RedHat 7.1 and interconnected by 100Mbps Fast Ethernet. Host A is a network traffic generator with a 633 MHz PIII and 128M RAM. Each protected server is a Pentium-Based PC with an 850 MHz CPU and 512M RAM.

Figure 10 illustrates test topology with gateway firewall, which is a PC with the same configuration as each protected server. Gateway firewall installed using IPTABLES package with overall policy in the filter list.

Figure 11 depicts experimental environment of distributed μFirewall architecture. In this architectural environment, Ethernet hub is connected directly to the outside world. Moreover, we eliminate the gateway machine. A policy administrator with an IRA is added to maintain consistent global security policy. Each of the server machines installs with a μFirewall and enforces policy locally.

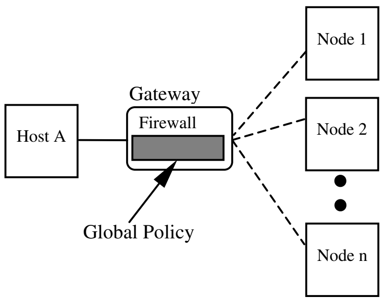


Fig. 10. Environment with CFW

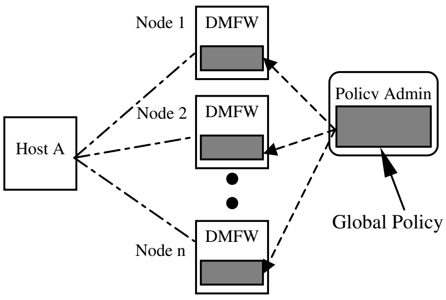


Fig. 11. Environment with DMFW

6.1 Security Evaluation

To evaluate security of a system, techniques designed to defeat and bypass the firewall are used to determine the effectiveness of the firewall. As a matter of fact, it is difficult to simulate a real attacker's behavior in a LAN experiment. However, the vulnerability of a specified firewall can be checked with network scanning tools. We use Nessus and other security tools for host attack and scanning.

Because security level of the firewall is directly related with policy configuration, we set the security policy in both CFW and DMFW as follows:

1) Deny any service request unless it is explicitly permitted;
2) Deny any ping packets from the outside.

In the centralized architecture, a gateway firewall is installed with a network-based intrusion detection system (NIDS, such as *snort*) to response some attack events. In our distributed approach, host-level IDA checks all attack events and reports them to IRA. A particular host attacks the *internal* nodes and computes success times. Here, the word *success* means that the attack behavior isn't denied by firewall. The results of these security experiments are summarized in Table1.

From Table1, we observe that DMFW is more secure than CFW when distributed attack (e.g., DDoS) or host-based attack (e.g., Password guessing) happens. Therefore, hierarchical intrusion response mechanism enhances the security capability of the protected system.

Table 1. Security Evaluation Results

No. of success count(s) / Attack Category/ No. of Attacks	CFW	DMFW
Port scan / 10	4	4
Syn-flooding Attack / 40	5	5
Fragmental Attack / 10	10	8
DDoS / 10	8	4
Password guessing / 40	30	20

6.2 Performance Evaluation

In the performance evaluation, we assume that every protected node insides a policy domain with same number of rules. We configure 4 internal servers to provide HTTP, FTP, TELNET, and SSH service, respectively. In our system, once IRA at policy administrator finds a severe intrusion event, it will add a counteract rule to all of internal servers. Hence, we can simply assume that the number of global policy rules is always four times that of each protected node. We also configure the gateway firewall with the same number of rule set as that of global policy in DMFW case. The experiments are done on the firewall to measure the performance degradation of two types of representative service, i.e. HTTP and FTP.

For HTTP session experiments, a particular host generates 1000 requests to a static web page and average session time is measured. In CFW, the gateway firewall controls network traffic and redirects legal packets to a web server. When we add the screening rules at the HTTP server in DMFW, the number of gateway firewall rules is augmented 4 times accordingly. Table 2 illustrates the experiments results. From the result, we see that gateway firewall architecture costs nearly double processing time compared to our architecture when the amount of HTTP-relevant rules reaches 300 or more.

In FTP experiments, a client running on a particular host connects to an internal FTP server and transfers a 44.1MB file. Table 3 shows the transaction latency time versus number of FTP-related rules.

Table 2. Average HTTP Latency (*ms*) vs. No. of HTTP-related Security Rules

	0	100	200	300	400	500	600
CFW	0.128	0.168	0.177	0.205	0.315	0.370	0.630
DMFW	0.128	0.142	0.149	0.154	0.164	0.169	0.173

Table 3. Average Latency (*ms*) vs. No. of FTP-related Security Rules for 44.1 MB Data

	0	100	200	300	400	500	600
CFW	89.719	89.993	90.170	91.159	91.513	105.762	115.112
DMFW	89.719	89.758	89.763	89.828	89.900	90.230	90.989

6.3 Scalability Evaluation

To demonstrate the scalability of the distributed μFirewall, we varied the number of hosts that participate in the connection setup. In this experiment, all of the internal nodes are running HTTP service. A particular host generates 800 connection requests to inside nodes. Gateway schedules the requests in a round-robin fashion. We first measure average transaction delay without firewall as the base line for comparison. In the experiment with gateway firewall, the gateway firewall filters illegal network traffic. In the DMFW experiment, packet filtering is done at the internal nodes. In our experiments, the number of global policy rules keeps 3200. In CFW case, these rules are preloaded at gateway firewall. In DMFW situation, the rules are distributed to the relevant nodes. We make the assumption that every protected node contributes roughly the same number of rules. Table4 depicts the processing overhead as the number of nodes increases. In Table 4, *insecure* means there is no firewall running.

Table 4. Average Transaction Delay (*ms*) with respective to Host Numbers

	1 Hosts	2 Hosts	3 Hosts	4 Hosts	5 Hosts	6 Hosts	7 Hosts	8 Hosts
CFW	3.517	3.450	3.400	3.300	3.200	3.100	3.000	2.800
DMFW	3.517	1.557	1.068	0.677	0.600	0.500	0.380	0.270
Insecure	0.550	0.450	0.400	0.350	0.300	0.250	0.200	0.150

In our system, policy cache mechanism enables each enforcement point keep a small subset of the whole rules. This experiment clearly demonstrates improvement in performance and the additional increased flexibility and scalability of policy caching and enforcing at the protected node.

7 Related Works

The distributed security architecture prompted by Kai Hwang [10] is quite similar to our work. In their design, gateway firewall, policy manager at the DMZ and micro-firewalls jointly provide a defense in depth in the protection of Intranet. Our approach is based on end-to-end security and is topology independent. Moreover, the *policy-updates on demand* mechanism reduces network traffic compared to their approach. Our preliminary results also indicate that fully distributed μFirewalls combining with DIDR system will improve security capability without performance degradation.

In the distributed security system, firewall management becomes a significant challenge. The administration of many independent firewalls is discussed by Miller [13]. It is argued that the firewalls should all be the same, though the management of different policies is not mentioned. Firmto [3] describes a toolkit for specifying policy independent of specific devices. In this approach, policy updates force complete loads of the rule-sets at the affected enforcement points. This causes scalability problems with respect to the number of policy entries.

Our work is also related to intrusion detection and response techniques. The Hummingbird system [9] designed by D.Frincke et al. is aimed to provide a cooperative intrusion detection framework. In their system, Kerberos is used for authentication of communication between hummers. In our system, however, KeyNote trust management system can provide end-to-end authentication comfortably and scale well to very large networks.

8 Conclusions and Future Works

This paper presents a viable end-to-end security architecture. In this architecture, security enforcements are pushed to the protected endpoints. At the endpoint, μFirewall is built with a packet filter based on DTE-enhanced OS and provides dual protection. The DIDR system provides hierarchical intrusion response capability. Mobile agents is used to accomplish fast policy delivery. Moreover, we exploit KeyNote trust management system to express AC, security policy and intrusion alarm messages.

Mobile-agent system saves network latency and bandwidth at the expense of imposing higher workload on their hosts and lower speed in policy distribution. Therefore, we plan to implement a more efficient and scalable mechanism for policy distribution, such as RMI broadcast and Reliable Multicast protocol which can provide greater scalability, security and lower latency. In the future, we will test our architecture with a larger scale deployment, validate intrusion detection and response on real traffic.

Acknowledgments

This research is supported by the Natural Science Foundation of Hubei Province under grant 2001ABA001.

References

1. Amoroso, E.: Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Traps, Trace Back, and Response, Intrusion.Net Books, 1999.
2. Badger, L., Sterne, D. F., Sherman, D. L., Walker, K. M.: A Domain and Type Enforcement UNIX Prototype, USENIX Computing Systems, Vol.9, Cambridge, Massachusetts, 1996.
3. Bartal, Y., Mayer, A., Nissim, K., Wool, A.: Firmato: a novel firewall management toolkit, Proceedings of IEEE Symposium on Security and Privacy, pp.17-31, 1999.
4. Bellovin, S. M.: Distributed Firewalls, login:, November 1999, pp.37-39.
5. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A. D.: The KeyNote trust management system, version 2•Internet RFC 2704•Sept. 1999.
6. Blaze, M., Ioannidis, J., Keromytis, A. D.: Trust management and network layer security protocols, Proceedings of Security Protocols International Wrokshop, Springer Verlag LNCS, 1999.
7. Chess, D.: Security Issues in Mobile Code Systems, Mobile Agent Security, Lecture Notes in Computer Science, Vol.1419, 1998, Springer, pp.1-14.
8. Clarke, D., Elien, J. E., Ellison, C., Fredette, M., Morcos, A., Rivest, R. L.: Certificate Chain Discovery in SPKI/SDSI, Technical Report, Computer Science Dept, MIT, November 1999.
9. Frincke, D., Tobin, D., McConnell, J., Marconi, J., Polla, D.: A Framework for Cooperative Intrusion Detection, Proceedings of the 21 st National Information Systems Security Conference, pp. 361-373, October 1998.
10. Hwang, K., Gangadhran, M.: Micro-Firewalls for Dynamic Network Security with Distributed Intrusion Detection, Proceedings of IEEE Int'l Symposium on Network Computing and Applications, June 20, 2001.
11. Ioannidis, S., Keromytis, A. D., Bellovin, S. M., Smith, J. M.: Implementing a Distributed Firewall, Proceedings of 7th ACM conference on Computer and Communication Security, Nov. 2000, Athens, Greece.
12. Keromytis, A. D.: STRONGMAN: A Scalable Solution to Trust Management in Networks, Ph.D. Thesis, University of Pennsylvania, November 2001.
13. Miller, M., Morris, J.: Centralized administration of distributed firewalls, Proceedings of Systems Administration Conference, pp.19 - 23, USENIX, 1996.

Encoding Function Pointers and Memory Arrangement Checking against Buffer Overflow Attack[⋆]

Changwoo Pyo[1] and Gyungho Lee[2]

[1] Hongik University, School of Information and Computer Engineering
72-1 Sangsudong, Mapo, Seoul 121-791, Korea
pyo@hongik.ac.kr

[2] University of Illinois at Chicago, Dept of Electrical and Computer Engineering
851 South Morgan St.,Chicago, IL 60607, USA
ghlee@ece.uic.edu

Abstract. Program counter is the only mechanism for processor to access instruction to execute. Protecting program counter is the fundamental defense for securing computer system. This paper presents a scheme of protecting program counter by encoding function pointers. In the scheme, every function address is encoded by linker. Compiler inserts instructions for decoding function addresses before call instruction. Encoding code pointers, function addresses as well as return addresses in stack frame, provides substantial coverage of protecting program counter. Several suggestions are also made to detect compromised code pointers at run-time without memory space for sensor mechanism. A demo Linux system has been under construction with the proposed scheme. Experimental data shows performance slowdown less than 10% when all return addresses and function addresses are encoded. With a Pentium III processor of 866MHz, the overhead for each function call is on the order of nanoseconds. We plan to migrate parts of our code pointer encoding scheme from linker to dynamic linker, which should improve security and performance.

Keywords: buffer overflow attack, program counter, function pointer encoding, memory arrangement checking

1 Introduction

The outbreak of Code Red Worm in July 2001 [11] testifies that buffer overflow attack is still prevalent. More than 1/4 million systems were infected in 9 hours according to the CERT advisory report [12]. Recent statistics since 1998 shows that almost 50% of attacks are related to buffer overflow [22]. The history of buffer overflow attack has been long and notorious as in Morris worm incident [13].

[⋆] This work was supported in part by NSF (CCR-0113409)

Attack stages	Countermeasures
	Prevention-oriented
Overflow and compromising a code pointer	
	Dynamic detection and recovery
Loading PC from the compromised pointer	
	Avoiding intrusion

Fig. 1. Progress of buffer overflow attack and countermeasures

Program counter (PC) is the only mechanism for processor to access instruction to execute. Protecting PC is the fundamental defense for securing computer system. The target of buffer overflow attack is PC. The attack proceeds in two stages [1,19]. First, a buffer is overflowed beyond its limit, and a code pointer becomes compromised. A return address pointer in stack frame or a function pointer is utilized. A code pointer can be compromised sometimes by dereferencing a data pointer compromised by buffer overflow. Second, PC is loaded from the compromised code pointer. Execution of a return or a call instruction loads PC from the compromised pointer.

Countermeasures against buffer overflow attack intervene at one of the three times as shown in Figure 1. *Prevention-oriented* solutions do not allow attacks to proceed into the first stage. Buffer overflow cannot occur if programs check bounds for all array and pointer accesses. However, bound-checking is usually omitted in C/C++ programming, since bound-checking imposes heavy performance overhead. It is reported that bound-checking may slow down matrix multiplication up to 30 times [15]. The overhead can be reduced if bound-checking would be applied selectively to vulnerable codes identified by static analyses [16,21,22]. If some vulnerable codes are missed and thus unpatched, there is no way of protecting the vulnerability at run-time.

Although a buffer is overflowed, an attack cannot win program control, unless PC is loaded from a compromised code pointer (progression into the second stage). *Dynamic detection* checks the integrity of code pointers at run-time when they are actually referenced [6,9]. Compilers install sensor mechanism. If a sensor detects a compromised code pointer at run-time, an exception handler is invoked for recovery. Sensor-based detection has been successful for protecting return address against linear stack smashing but not effective against non-linear attack. Code pointers can be compromised without touching sensors [4,7,23]. Protecting function pointers is another unresolved problem. The fundamental limitation is that sensor data structures in writable memory are *yet another vulnerable locations.*

Intrusion can be *avoided* even when PC is loaded from a compromised code pointer at the second stage. Suppose all legitimate programs follow a private protocol, for example *program counter encoding* (PC-encoding for short) [17]. The protocol encodes and decodes an address when it is stored into a code pointer and loaded into PC respectively. Attackers may compromise the code pointer by buffer overflow without proper encoding prior to loading PC, but the decoding would produce an erroneous address. If the attacked program is equipped with

```
#include <stdio.h>

char buf[10];
void (*fp)();  /* vulnerable function pointer */

void func() { printf("%s\n", buf); }

int main() {
  fp = func;
  gets(buf);
  func();       /* plain function call */
  fp();         /* call via a function pointer */
  return 0;
}
```

Fig. 2. Buffer overflow vulnerability by a function pointer

an appropriate exception handler, it can nullify the attack and revive itself. PC-encoding integrates security mechanism with the object to be protected (i.e. PC), where prevention- and detection-oriented approaches encapsulate the object with shield. Since PC is the final front presiding execution, protecting objects other than PC with separate security mechanism is hardly free from vulnerability.

There has been no satisfactory solution against attacks on function pointers other than bound-checking. Figure 2 illustrates the problem. The function pointer `fp` is compromised by overflowing the buffer `buf`. Program control can be hijacked when the function `func` is called by way of the pointer `fp`. It makes no difference whether the variables `fp` and `buf` are defined inside the function `main` or outside.

This paper presents a scheme of protecting PC by encoding function pointers. In the scheme, every function address is encoded by linker. Compiler inserts instructions for decoding function addresses before call instruction. Encoding function addresses complements PC-encoding [17] originally developed for protecting return addresses in stack frame. Encoding code pointers, function addresses as well as return addresses in stack frame, provides substantial coverage of protecting PC. The remaining code pointers to be protected are those in the buffer structures used by `setjmp` and `longjmp`. Several suggestions are also made to detect compromised code pointers at run-time without memory space for sensor mechanism. They are collectively called *memory arrangement checking.*

We review the software solutions for buffer overflow attack in Section 2. Protecting code pointers by encoding is discussed with emphasis on function pointers in Section 3. Section 4 presents memory arrangement checking. Experimental performance data is shown in Section 5. Section 6 discusses several improvements on our current implementation and wraps up this paper.

2 Related Work

Traditional program analysis and transformation for pointer and array bound-checking [2,15] can be used to prevent buffer overflow as well as to reduce runtime errors. Integer range analysis [22], annotation-assisted analysis [16] and scanning for functions known to be vulnerable [21] belong to specialized static analyses to accurately locate vulnerable codes. The identified vulnerable codes are patched manually or automatically.

StackGuard, a pioneering work by Cowan *etal.* [9], represents dynamic detection. StackGuard puts a *canary word* as a security sensor data structure on top of a return address when a function is called. Before a return instruction is executed, a compromised return address is detected by comparing the canary's value with its original one. It is assumed that a return address pointer cannot be compromised without overwriting the canary sensor during buffer overflow attack. Compared to rigorous bound checking, StackGuard has superior performance.

StackGuard has several shortcomings. StackGuard cannot detect non-linear buffer overflow attack [4,7,23]. The change in the layout of stack frame by canary may cause compatibility problem. It is no longer cost-effective to protect function pointers in the framework of StackGuard. Suppose a function address is copied from a function pointer to another. To make it worse, a function address can be copied from a function pointer to an integer variable through type casting. There would be not a few locations to protect with the inbetween sensors. An extension of StackGuard, *PointGuard* is proposed to protect function pointers [10], but no report is available yet.

Return address defender (RAD) [6] is another dynamic detection. It saves copies of return addresses in a separate segment so that a return address in a stack frame can be compared with the corresponding one in the reservoir when return instruction is executed. It has advantage of preserving stack frame layout. The reservoir's integrity can be checked by parenthesizing it with sensors similar to canary, but it is not free from the yet-another-vulnerable-location problem. The problem may be avoided by toggling access right between writable and non-writable states whenever necessary, but its overhead becomes unbearable. RAD has no solutions for function pointers yet.

Libsafe [3] replaces vulnerable libraries with safe ones without disturbing existing binaries that depend on the libraries. It intercepts vulnerable function calls and redirects them to the corresponding safe ones. Recompiling the libraries and programs using them are not required. They also show that return addresses can be checked in existing binaries by using binary editing technique (Libverify). Linkers and dynamic linkers take an active role in installing detective mechanism in binaries. Libsafe can be classified as preventive, and Libverify, as dynamic detection.

PC-encoding [17] does not require bound-checking and dynamic detection. When a function is called, the return address is encoded before stored in stack frame. The encoded return address is decoded before returning to the calling function. Any function call and return not following encoding and decoding with a proper key cannot continue normal execution. Since PC-encoding does not perform any dynamic detection, it has little overhead compared to other work. PC-encoding can be implemented at architecture or compiler level. PC-

```
<main>:
xor    %esp,(%esp,1)         /* prologue: encoding return address */
push   %ebp                  /* prologue: save frame pointer */
mov    %esp,%ebp             /* prologue: new frame pointer */
sub    $0x14,%esp            /* prologue: adjust stack pointer */
push   %ebx
movl   $0x3b37f333,0x8049658 /* fp = func */
add    $0xfffffff4,%esp
push   $0x804964c
mov    $0x3b37f47f,%ebx      /* load encoded address of gets */
mov    0x8049558,%eax        /* load the key */
xor    %eax,%ebx             /* decode */
call   *%ebx                 /* gets(buf) */
add    $0x10,%esp
mov    $0x3b37f333,%ebx      /* load encoded address of func into ebx */
mov    0x8049558,%eax        /* load the key */
xor    %eax,%ebx             /* decode */
call   *%ebx                 /* func(); */
mov    0x8049658,%ebx        /* load %ebx from fp */
mov    0x8049558,%eax        /* load the key */
xor    %eax,%ebx             /* decode */
call   *%ebx                 /* fp(); */
mov    $0x1,%eax
jmp    80484cc <main+0x54>
lea    0x0(%esi),%esi
mov    0xffffffe8(%ebp),%ebx
mov    %ebp,%esp             /* epilogue: restore stack pointer */
pop    %ebp                  /* epilogue: restore frame pointer */
xor    %esp,(%esp,1)         /* epilogue: decode return address */
ret
```

Fig. 3. Encoded function calls

encoding should be extended to protect function pointers in addition to return address pointers.

3 Protecting Program Counter by Encoding

We briefly review return address encoding before discussing function address encoding. In our current implementation for x86, return addresses are encoded and decoded by `xor`'ing with the stack pointer register (`%esp`). Since two additional `xor` instructions are required for encoding and decoding, the overhead is minimal. Called functions are responsible for the encoding and decoding protocol. The corresponding calling functions are not aware of the enforcement of the protocol. See the function prologue and epilogue in Figure 3. Given a return address v, it is kept as v `xor %esp`. The encoded return address v `xor %esp` is decoded in function epilogue before executing a return instruction. Applying `xor` again gives the return address (v `xor %esp`) `xor %esp`, which is equal to v.

When PC is loaded with a value from text segments, there is no chance of being compromised. It is unnecessary to encode function addresses appearing in text segments. The address of the function `func` appearing in the call with comment "`/* plain function call */`" in Figure 2 does not have to be encoded. The assignment `fp = func` moves the address of the function `func` to the function pointer `fp` in data segment, which is writable and thus vulnerable. The address stored in `fp` should be encoded. If the value in `fp` was encoded, it should be decoded before the following call `fp()` is executed. Though it may produce efficient programs to encode function addresses selectively depending on which segments they are from, function addresses are indiscriminately encoded for easy implementation.

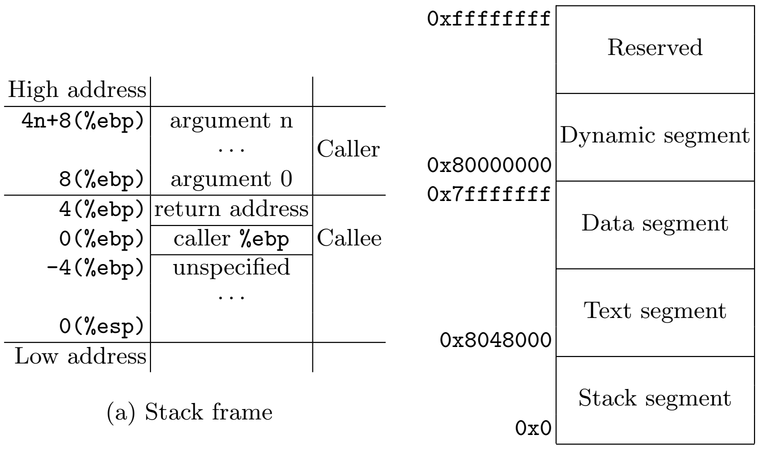
Since function addresses are not fixed until relocation is completed, linker encodes function addresses during relocation. When linker relocates a global symbol, if the type of the symbol represents a function, linker encodes the address resulting from relocation. We call it *encoding relocation.* A randomly generated key is `xor`'ed with the relocated function address. The encoding key is stored in a global variable so that it can be used for decoding at run-time. This key management bears some vulnerability including the yet-another-vulnerable-location problem, but this can be overcome by dynamic linker (See Section 6).

It is compiler's responsibility to generate instructions with anticipation of linker's encoding relocation. Since linker performs encoding relocation independently of involvement of a function pointer, decoding instructions should be placed before every call. For a call to a function `f`, compiler generates the following instructions:

1. load `f`'s address into a general register r
2. load the decoding key into a register k
3. `xor` the address in r by using k (decoding)
4. call using the register r, which now holds valid `f`'s address.

Figure 3 shows the machine code for x86 processors translated from the program in Figure 2. The addresses of functions `gets` and `func` appear as encoded. The encoded address of function `func` is copied to function pointer `fp` as usual. Since all function addresses are encoded, they are decoded when call instructions are executed.

When a program with PC-encoding is attacked by buffer overflow, the outcome would be similar to that of the denial-of-service attack [5] in our current implementation. Although the program would not allow intrusion, it would be hung up. If attackers' purpose is to crash the target program or system, winners are the attackers. However, they cannot succeed in intruding the system under attack. This suicidal defense is useful for blocking self-replicating worms such as Code Red. PC-encoding can be used independently of other measures. Any attack missed by bound-checking and dynamic detection still has to face up to the combinatorial complexity of guessing correct key to break in a system alive.



(a) Stack frame

(b) Segment placement

Fig. 4. Memory arrangement for x86 microprocessor

4 Memory Arrangement Checking

Memory arrangement checking requires no additional memory space for security sensor. Instead, memory arrangement checking inspects stack pointer, frame pointer, segment bounds and instruction alignment. Since the original memory layout is preserved, there is no compatibility problem. It can be used freely with other solutions to reinforce self-protecting programs, for example, those using PC-encoding. We suggest three methods of memory arrangement checking. They are not comprehensive by any means but as powerful as canary by StackGuard. They are explained in the context of x86 processors with System V application binary interface [20].

4.1 Frame Inversion

Since stack grows downward from high to low memory, stack pointer register `%esp` should always have values smaller than those of frame pointer register `%ebp`. If `%ebp - %esp < 0`, then the stack frame is *inverted.* This is effective for detecting linear stack smashing attack.

Figure 3 shows also the typical prologue and epilogue code sequences. Stack frame is shown in Figure 4(a). A call instruction pushes a return address into the stack. Called function pushes caller's frame pointer into the stack, and copy the stack pointer's value to the frame pointer. Before a return instruction is executed, the old frame pointer value is popped into the frame pointer register. The caller's stack frame is recovered. Return address is now at the top of the stack. The return instruction, popping the return address into PC (instruction pointer register for x86 processor), transfers control to the instruction following the call instruction.

Caller's stack frame is inverted if a return address pointer is smashed from below. Since attack code is injected into the buffer in the called function's stack frame, the injected attack code should have address lower than the frame pointer's value. If the return address pointer is smashed from below, the caller's frame pointer slot on top of the return address pointer should also be compromised. Therefore linear attack inverts caller's stack frame.

4.2 Destination Segment

This is a very crude bound checking. The 32-bit virtual address space for x86 is divided into 5 non-overlapping segments. They are reserved, dynamic, data, text, and stack segments from high memory. Figure 4(b) shows a typical segments arrangement. Any call or return into non-text segment may be regarded as intrusion. If segment checking would be used with PC-encoding, we only encode and decode lower order bits. The size of text segment determines the number of lower order bits subject to encoding. Higher order bits necessary to distinguish segments are untouched. Higher order bits of a compromised code pointer would address a segment other than text. In case of x86, if the higher order 20-bit chunk of an address is less than `0x08048`, then the address belongs to stack segment.

4.3 Address Rotation

This method is effective for RISC processors, where every instruction is of the same length, say four bytes, and alignment is compulsory. If instructions should be aligned at the multiple of four, the two least significant bits of every instruction's address are always 0. Address encoding proceeds in two steps. Address is `xor`'ed with a key, which puts 2-bit mark 00, 01, 10, or 11 into the two least significant bit positions. The resulting bits are rotated by n positions, where n is less than the size of an instruction.

The integrity of the encoded address is checked in reverse order. First, address bits are rotated by n bits in opposite direction. Then the 2-bit mark in the least significant bit positions are checked. If the mark is correct, the address is decoded by `xor` instruction. The 2-bit mark and the number of bit positions to rotate can be chosen randomly for each execution if dynamic linker takes the responsibility of encoding.

Address rotation can be applied to microprocessors that do not enforce alignment on instructions such as x86. Compiler can translate every return address to be a multiple of some 2's power. For example, if return address should be a multiple of 8 bytes, we have three bits to store marks in. Selecting one out of the eight 3-bit marks, `xor`'ing and rotating complete encoding.

5 Experiment

A demo Linux system has been under construction with the proposed scheme. We rebuilt gcc version 2.95 for encoding return addresses. Using the compiler, Linux

Table 1. Execution times and code sizes when gcc and its variations are used

	Execution time [sec]			Code size [byte]		
Programs	gcc	+RAE	+FPE	gcc	+RAE	+FPE
`inc`	19.77	23.55 (0.84)	23.63 (0.84)	13,504	13,556 (1.00)	13,568 (1.00)
`compress`	4.19	4.37 (0.96)	4.54 (0.92)	87,717	87,941 (1.00)	88,658 (0.99)
`hanoi`	29.27	33.92 (0.86)	37.80 (0.77)	13,573	13,625 (1.00)	13,677 (0.99)
`quicksort`	0.72	0.75 (0.96)	0.77 (0.94)	14,173	14,241 (1.00)	14,333 (0.99)
`fibonacci`	5.24	5.85 (0.90)	7.38 (0.71)	13,630	13,698 (1.00)	13,734 (0.99)

Table 2. Time overhead for a function call by PC-encoding

		Overhead/call (sec)	
Program	Number of calls	+RAE	+FPE
`inc`	1,000,000,000	$3.780 * 10^{-9}$	$3.860 * 10^{-9}$
`compress`	72,828,619	$2.472 * 10^{-9}$	$4.806 * 10^{-9}$
`hanoi`	1,073,741,823	$4.331 * 10^{-10}$	$7.944 * 10^{-10}$
`quicksort`	9,480,029	$3.165 * 10^{-9}$	$5.274 * 10^{-9}$
`fibonacci`	204,668,309	$2.980 * 10^{-9}$	$1.046 * 10^{-8}$

system kernel version 2.4, major libraries and Apache server are recompiled. The rebuilt system is used as our test bed. We rebuilt another version of compiler and the linker in binutility version 2.10 for encoding function addresses. Currently it works for only non-position-independent codes (non-PIC).

Table 1 shows the execution times and code sizes of five programs with many function calls. They were compiled by using three different compilers, and executed by Pentium III 866 MHz with Linux system (kernel version 2.4). The program `inc` calls the function `int inc( int i ) { return ++i; }` 1 billion times. The `compress` is from the SPEC 95 integer benchmark suite. Rest of the tested programs are heavily recursive so that we can magnify the overhead inflicted by PC-encoding. Program `hanoi` (Tower of Hanoi) moves 30 disks, `quicksort` works on 1 million numbers and `fibonacci` computes 40th Fibonacci number. The second and the fifth columns show the execution times and the code sizes when the original gcc compiler is used, the third and the sixth, when gcc with return address encoding (RAE) is used, and the fourth and the seventh, when gcc with both of return address and function pointer encoding (FPE) is used. The numbers in parentheses are the slowdown for execution time, which is execution time before PC-encoding divided by execution time after PC-encoding. Code size change is computed similarly. Note that the programs `compress` and `quicksort` show slowdown less than 0.1 and the code sizes change little.

Table 2 show the time overhead for each function call calculated from the Table 1. The difference in execution times is divided by the the number of calls. We observe that the overhead for a call is on the order of nanoseconds, which are negligible.

Table 3 shows the result of benchmarking Apache server using WebStone 2.5 [18]. Apache server's platform was Pentium 450Mhz with Linux with kernel

Table 3. Performance of Apache web server with return address encoding

RAE	Num. of clients	Conn. / sec	Avg. latency (sec)	Avg. throughput (M bits/sec)
no	2	48.03	0.042	7.52
no	16	49.21	0.325	7.57
no	30	49.66	0.601	7.73
yes	2	47.94	0.042	7.51
yes	16	45.81	0.348	7.01
yes	30	49.13	0.608	7.62

2.2.16 and the client's was Sun UltraSPARC. Linux kernel and Apache are rebuilt by return address encoding gcc. Other experimental parameters are the same as those in [8].

The results show that PC-encoding burdens system with little overhead for protecting program counter. For return address protection, we need only two instructions for encoding and decoding. For a function call including calls using function pointers, we need additional two instructions for loading encoded address and key, and one instruction for decoding. The overhead would be approximately proportional to the number of these additional instructions.

6 Discussion and Conclusion

We have demonstrated how function pointers can be encoded by making compiler and linker work together. When compiler translates a function call, instructions are generated in anticipation of linker's encoding relocations. Linker encodes the function address for which compiler have made provision by generating decoding instructions. Experimental data shows performance slowdown less than 10% when all return addresses and function addresses are encoded. With a Pentium III processor of 866MHz, the overhead for each function call is on the order of nanoseconds.

Although it is hard for buffer overflow attack to guess the key for encoding and break in a system, a program with PC-encoding would fault or crash when attacked in our current implementation. To overcome the problem, memory arrangement checking can be employed. In particular, address rotation is promising, since it checks integrity of encoded addresses by themselves echoing to the PC-encoding's self-protection philosophy. PC-encoding with memory arrangement checking can protect programs at low cost.

Our implementation still needs improvements. Dynamically varying keys should be used for encoding. This means function addresses should be encoded differently for each execution. It is also desirable for the key to differ for each function. Current implementation relies on static key, which is the same for all functions and invariable unless linked again. This problem can be solved by letting dynamic linker take the encoding responsibility. Whenever a program is executed, dynamic linker generates a random key and perform encoding relocation for each function symbol at run-time. It is possible to encode each function address differently if dynamic linker keeps track of function-key pairs. It is necessary to develop new relocation types to solve the problem systematically with a view from dynamic linker. This scheme can also be applied to return address encoding.

Keys should be kept in non-attackable locations. In our current implementation, linker and compiler exchange a key through a global location in data segment, which is definitely a vulnerable location. Text segment is the best place to keep the key in the current architecture, since it is write-protected. Keys can be represented as immediate operands of some instructions. Dynamic linker can patch key values directly in text segment at the beginning of execution.

Our PC-encoding has a limitation when type casting gets involved. Function pointers can be cast to integer type in ISO C [14]. Thereafter it can be cast to any other type. This includes casting back to the original function pointer type. When an encoded function pointer is cast to an integer, its value can be changed. Casting the integer back to the function pointer followed by decoding, no one can expect what it would be. Although this kind of programming is believed not to be used widely, the language definition allows it. Our PC-encoding, in particular, function pointer encoding is no longer viable under this situation.

While we were working on function pointer encoding, Linux kernel and Apache server rebuilt with return address encoding have been serving for almost a year without any performance degradation and significant errors. PC-encoding compiler with intrusion detection by memory arrangement checking makes a successful self-protecting solution against the infamous buffer overflow attacks. For more secure protection against more sophisticated future attacks that may compromise function pointers for position independent code (PIC), our solution needs more work because PIC requires dynamic linker. We plan to migrate parts of our code pointer encoding scheme from linker to dynamic linker, which should improve security and performance.

References

1. Aleph One. Smashing The Stack For Fun And Profit. *Phrack Magazine*, 7(49):File 14, 1996.
2. Todd M. Austin, Scott E. Breach, and Gurindar S. Sohi. Efficient Detection of All Pointer and Array Access Errors. In *Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation*, 1994.
3. Arash Baratloo, Navjot Singh, and Timothy Tsai. Transparent Run-Time Defense Against Stack Smashing Attacks. In *Proceedings of the USENIX Annual Technical Conference*, June 2000.
4. Bulba and Kil3r. BYPASSING STACKGUARD AND STACKSHIELD. *Phrack Magazine*, 10(56), May 2000.
5. CERT Coordination Center. CERT Advisory CA-1996-26: Deanial-of-Service Attack via ping. http://www.cert.org/advisories/CA-1996-26.html, Dec 1996.
6. Tzi-Cker Chiueh and Fu-Hau Hsu. RAD: A Compile-Time Solution to Buffer Overflow Attacks. In *21st International Conference on Distributed Computing Systems*, 2001.

7. CORE SECURITY TECHNOLOGIES Inc. Multiple Vulnerabilities in Stack Smashing Protection Technologies. *SecurityFocus Online*, Apr. 2002.
8. Crispin Cowan, Steve Beattie, Ryan Finnin Day, Calton Pu, Perry Wagle, and Erik Walthinsen. Protecting Systems from Stack Smashing Attacks with StackGuard. In *the Linux Expo*, 1999.
9. Crispin Cowan, Calton Pu, David Maier, Heather Hinton, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. StackGuard: Automatic Detection and Prevention of Buffer-Overflow Attacks. In *the 7th USENIX Security Symposium*, 1998.
10. Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. In *the DARPA Information Survivability Conference and Expo DISCEX*, 1999.
11. Roman Danyliw and Allen Householder. CERT Advisory CA-2001-19: Code Red Worm Exploiting Buffer Overflow In IIS Indexing Service DLL. http://www.cert.org/advisories/CA-2001-19.html, Jul 2001.
12. Roman Danyliw and Allen Householder. CERT Advisory CA-2001-23: Continued Threat of the Code Red Worm. http://www.cert.org/advisories/CA-2001-23.html, Jul 2001.
13. M. W. Eichin and J. A. Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. In *Proc. IEEE Symposium on Security and Privacy*, pages 326–343, 1989.
14. Samuel P. Harbison and Guy L. Steele Jr. *C, A Reference Manual.* Prentice Hall, 4th edition, 1995.
15. R. W. M. Jones and P. H. J. Kelly. Backwards-compatible bounds checking for arrays and pointers in C programs. In *Proceedings of the Third International Workshop on Automated Debugging*, 1997.
16. David Larochelle and David Evans. Statically Detecting Likely Buffer Overflow Vulnerabilities. In *2001 USENIX Security Symposium*, 2001.
17. Gyungho Lee and Akhilesh Tyagi. Encoded Program Counter: Self-Protection from Buffer Overflow. In *International Conference on Internet Computing*, Jun. 2000.
18. Mindcraft. Webstone standard web server benchmark. http://www.mindcraft.com/webstone.
19. Mudge. How to Write Buffer Overflows. http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html, 1995.
20. Unix System Laboratories. *System V Application Binary Interface: Intel386 Architecture Processor Supplement.* Prentice Hall, 3rd edition, Jan 1994.
21. J. Viega, J. T. Bloch, T. Kohno, and G. McGraw. ITS4: A Static Vulnerability Scanner for C and C++ Code. In *Proceedings of Annual Computer Security Applications Conference*, Dec 2000.
22. David Wagner, Jeffrey S. Foster, Eric A. Brewer, and Alexander Aiken. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. In *Network and Distributed System Security Symposium*, Feb 2000.
23. Mariusz Woloszyn. StackGuard Mechanism: Emsi's Vulnerability. http://immunix.org/StackGuard/emi_vuln.html, 1999.

An Evaluation of Different IP Traceback Approaches

Vadim Kuznetsov, Helena Sandström, and Andrei Simkin

Department of Computer Science and Electrical Engineering
Luleå University of Technology, SE-971 87 Luleå Sweden
{vadim,Helena.Sandstrom}@cdt.luth.se, anesim-8@sm.luth.se

Abstract. The problem of identifying the sources of a denial of service attack is among the hardest in the Internet security area, especially since attackers often use incorrect, or spoofed, source IP addresses. In this paper we present the results from a comparison between some of the most promising traceback techniques proposed to solve this problem. Our goal was to evaluate and analyze the most promising techniques on our way to find a more efficient approach. We have evaluated four different traceback approaches and summarized the results. Our own research was primary targeted at the iTrace approaches while the other approaches were evaluated based on the previous work. We conclude that there are two main disadvantages of the proposed approaches. First, the hop-by-hop path reconstruction is inefficient due to a significant computation overhead, or a long time spent for collecting the samples of the path. Second, the path reconstruction requires changes in the core routing structure that is not profitable. We also suggest a slightly modified version of iTrace approach, which aims at reducing the overhead imposed by such changes.

1 Introduction

Denial-of-service (DOS) attacks are a pressing problem in today's Internet. The impact caused is often more serious than network congestion because of their target specific and concentrated nature. In a *distributed* DOS (DDOS) attack, the attacker uses a number of compromised slaves to increase the transmission power and orchestrate a coordinated flooding attack. Highly automated attack tools have been developed where a common ingredient is the use of spoofed source addresses. Particularly, DDOS attacks with hundreds or thousands of compromised hosts, often residing on different networks, may lead to the target system overload and crash. Due to the stateless nature of the Internet, the dilution of locality in the flooding stream combined with spoofed source addresses undermines the effectiveness of traceback techniques for locating the sources. By the use of IP spoofing, stepping stone techniques, and zombie slaves, attackers can quite easily hide their identity. Therefore, finding the true identity of an attacker includes many steps, of which tracing the machines that directly generates the attack packets really is only the first step.

Another way to render efficient DDOS attacks, which do not include the use of compromised slaves, is by bouncing flooding traffic off of *reflectors*. Attackers can thereby effectively hide their own location. In this scenario the attacker sends a false request to the reflector on behalf of the victim, which is done by setting the victim IP address as the source address. By spoofing requests from the victim to a large set of

Internet servers (for example DNS, Gnutella, and web servers) attackers can make it really difficult for a victim to isolate the attack traffic in order to block it. As pointed out in [11] it proves impractical to defeat *reflector attacks* with traceback techniques, especially since the source addresses that reach a victim are the true IP address of each reflector. Therefore reflector attacks are not particularly considered in the context of traceback.

During recent years the problem with DOS attacks has become well known, but it has been difficult to find a way to measure the denial of service activity in the Internet as a whole. As far as we know there is only one publication in this area [4]. In this publication they have been able to measure, with some reliability, the activity of DOS attacks (excluding reflector attacks).

The experimental measurements give a conservative lower bound of approximately 20-40 attacks per hour in the whole Internet, directed to different network prefixes, with the mean duration of 10-15 minutes. If these results are correct, they give us a very sad imagination of today's Internet security.

The problem of tracing streams of spoofed packets has received considerable attention recently, and several approaches have been introduced in the Internet society. One technique is Ingress filtering as described in RFC 2827 [13]. The idea with ingress filtering is that packets from an edge network should be filtered using the prefix for that network. Any packet carrying an IP address with a wrong prefix should be blocked at the filtering router and not allowed to continue towards its destination. If ingress filtering could be implemented everywhere tracing traffic would be unnecessary. The problem is that it is not likely that all edge networks will ever implement this even though it is in use today. Therefore our belief is that it will still be interesting to find efficient traceback techniques.

The rest of this paper is organized as follows. Section 2 gives a short description of proposed techniques. In Section 3 parameters used for comparison have been introduced. Section 4 presents a modified iTrace approach and Section 5 describes the result. Finally we summarize in Section 6, and conclude in Section 7.

2 Traceback Approaches

The proposed traceback techniques can be sorted into three distinct categories:

1. Actively querying routers about traffic they forward.
2. Creating a virtually overlay network for selective monitoring of packet flows.
3. Identifying the attack path by reconstruction, using a collection of packets, marked or especially generated by routers along the attack path.

A simple scheme of the first category is in use today. If a victim recognizes that it is being attacked it develops an attack signature, consisting of some data common and unique to the attack traffic. A query including the attack signature is then sent hop-by-hop to each router along the path. This presuppose that each routing device supports input debugging and is able to tell about interface through which a packet corresponding to the attack signature has arrived. This technique is however not very efficient and requires a lot of manpower and good contacts with other network providers. Some ISP's may have implemented a more sophisticated and automated technique for this to speed up the trace procedure within their own network. A drawback of all techniques in the first category is that tracing can only be done during an ongoing attack.

In the second category we have placed different logging techniques [8, 12]. In general it is not feasible to use logging since it often requires huge storage capacities, but within this category there is one very interesting approach, Hash-Based IP traceback [8], more common called *SPIE* (Source Path Isolation Engine). With the use of an efficient logging technique, only collecting hashes of the packets, it is possible to trace a single route of one packet.

The third category includes different variants of probabilistic packet marking (*PPM*), first proposed by Savage and colleagues [2], and ICMP traceback (*iTrace*), first proposed by Bellovin and colleagues [5].

The basic idea behind *PPM* is the use of edge sampling. A packet on the path is with a certain probability marked by two routers on the way, forming an edge. Each marked packet then represents a sample of the whole path. The victim receives all packets and can thereby use the marked packet to reconstruct the entire path back to the source. Song and Perrig [3] have presented an enhanced version of PPM. They noticed that if the victim knows the map of its upstream routers the computation and reconstruction can be done much more efficient.

The basic idea behind iTrace is that every router should sample a packet with some probability, copy its content onto a special ICMP packet, add information about the adjacent upstream and/or downstream routers and send it towards the same destination as the original packet. The victim of an attack can then use these packets to reconstruct the paths back to the attackers. An observation made of this solution was that it is much more likely that the victim will get iTrace packets from routers nearby than from routers far away. A variant of iTrace, called intention-driven iTrace [6, 7] propose a solution to this, which increase the probability of receiving an iTrace message when needed. By using a special intention value that can be propagated to routers through BGP updates it is possible for a host or victim to raise the probability of receiving iTrace packets from remote routers.

A common disadvantage of the first category is that tracing cannot be done post mortem, after an attack has stopped. Within the second category, all techniques expect SPIE requires huge storage capacity. Even though SPIE is a promising technique, and is unique in that it can trace a single packet, it suffers from a very tight time limit and costly investments. Therefore we have chosen to focus on the various techniques proposed within the third category.

3 Evaluation of the Suggested Approaches

We have simulated the different traceback approaches on a linear, star, and tree topology. The results of the simulations were verified and compared to the results of mathematical analysis, where possible.

We have selected four distinct parameters for comparison and evaluation of different traceback approaches,

1. The number of packets required for a complete path reconstruction
2. The computation overhead of a reconstruction procedure
3. The robustness of a traceback mechanism in case of a large-scale DDOS
4. The deployment overhead

The first parameter represents estimated time for collecting enough packets to get required information for the reconstruction. The second parameter represents estimated time required for processing collected information to be able to reconstruct the path back to the attackers. It is desirable to minimize these two characteristics to achieve a fast response to an attack, and diminish the damage. The third parameter, the definition of robustness, was first given by Song and Perrig [3]. They introduced two terms: *false positive*, a path that do not take part in an attack but it is reconstructed by a traceback mechanism, and *false negative*, a path that take part in an attack but it is not reconstructed. The traceback approach is robust if it gives a relatively low rate of false positives and false negatives, and if the rate does not grow rapidly with increased number of attackers. The fourth parameter, the deployment overhead, cannot be measured or calculated directly but can be evaluated by means of common sense.

The choice of these parameters is motivated by our aspiration for a solution to the IP traceback problem that is time and cost efficient, and gives a high precision.

3.1 Methodology

Our evaluation is based on the contribution of previously published papers [2, 3, 7], simulations in the network simulator [10] and mathematical analysis.

The network simulator was extended to be used with both the PPM methods as well as with the two approaches of iTrace. The attack traffic was simulated by the UDP agents attached to every attack node and which were configured to send traffic with a rate of about 500 pkt/s. The connecting links between routers were configured to propagate data with high speed and minimal loss-rate in the queues (this was achieved by setting up high bandwidth of the links). A victim node collected all required data and stored it into a file. These data were further processed in matlab. The goal of the simulations was to evaluate statistically the behavior of the chosen approaches therefore every simulation with the same configuration was performed 100 times. In our simulation we used different topologies: linear topology, star and tree topology. In case of a tree topology a victim was located at the root of a balanced tree while the attackers were at the leaves of the tree.

3.2 Number of Packets Required for Reconstruction

The purpose of path reconstruction is to find the IP address of an attacking host or at least the address of the router closest to the attacking host. During the reconstruction procedure we have to verify that the reconstruction path is complete and correct. Neither PPM nor iTrace approaches provide a mechanism for verifying the completeness of the reconstructed path. The only way to verify this is to collect accordingly large amount of samples, and to make sure that there are no new samples.

The edge-marking algorithm in PPM depends primarily on the marking probability p. This parameter defines the fraction of packets, which are marked by a router. The expected number of collected packets required for path reconstruction for a linear topology is bounded by:

$$E\{N\} = \frac{k \cdot \ln(k \cdot d)}{p(1-p)^{d-1}} \tag{1}$$

where k is the number of samples required for reconstruction of a single edge between two routers and d is the length of the path from the victim to the attacker [2].

The method suggested by Savage and colleagues assumes k=8.Figure 1 represents the graph of this equation.

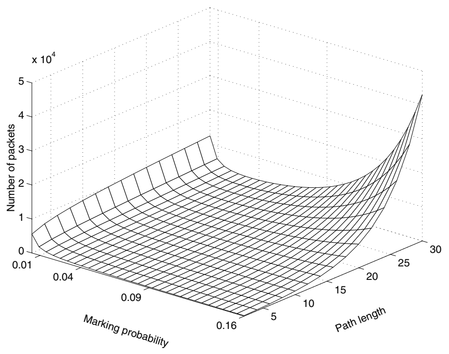


Fig. 1. The expected number of packets required for path reconstruction in Savage and colleagues algorithm

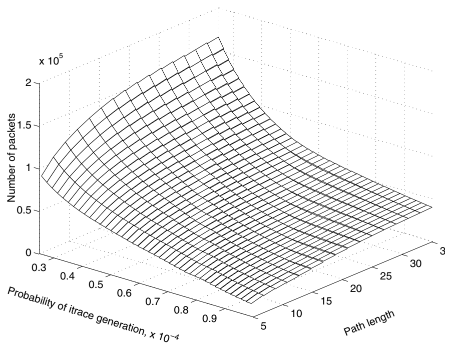


Fig. 2. The expected number of packets required for path reconstruction in iTrace

As Figure 1 indicates, with a value of p around 4-5%, the number of packets required for path reconstruction is minimized, and does not exceed $4 \cdot 10^3$ for all values of d in the interval between 2 and 32. It is remarkable that this number neither depends on topology nor the number of attackers. Assuming that the rate of emitting attack packets is equal to 10 packets per second, per attacking host (which is an extremely low rate [4]), we yield that the attacking hosts have to generate traffic as long as 5-7 minutes before a victim is able to reveal the whole path.

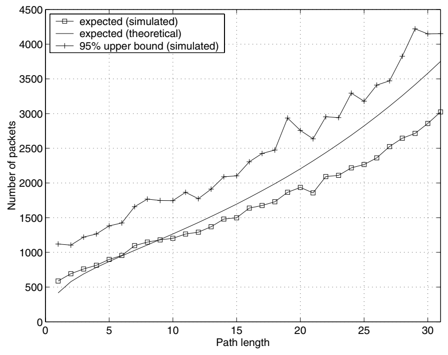
The method proposed by Song and Perrig requires fewer packets although the robustness suffers in this case [3]. The iTrace approach depends on the ICMP packet generating probability μ , which indicates the fraction of the packets being traced by a particular router. If, for example, this parameter is equal to 1/20000, it means that one packet out of 20000 could be traced and an ICMP packet containing information about the route of this packet is issued towards the destination. The reconstruction of the path from a victim to an attacker is possible if all the routers have generated at least one ICMP packet. We have found an appropriate formula showing the expected number of packets required to be generated by an attacking host (in case of linear topology), in order to make sure that all routers along the path have generated at least one ICMP packet.[1]

$$E\{N\} = \frac{\ln(d-1) + 0.58}{\mu} \tag{2}$$

where d is the length of the path from the victim to the attacking host Figure 2 represents the graph of this equation.

[1] The calculations are available upon request

Figures 3 and 4 show the dependency on a distance from an attacker to a victim for the particular values of parameters *p* (for Savage and colleagues) and μ (for iTrace) correspondingly. These figures show also a good correlation of the theoretical and simulated results.



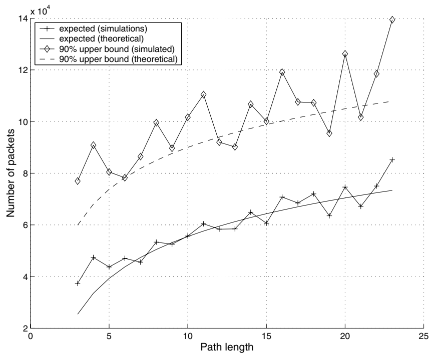


Fig. 3. The expected number of packets required for path reconstruction in Savage and colleagues algorithm in case of a linear topology (*p*=0.04)

Fig. 4. The expected number of attacking packets required for path reconstruction in iTrace in case of a linear topology ($\mu = 1/2^{15}$)

For the proposed value of μ =1/20000 and the mean Internet diameter 20 hops the whole path can be reconstructed after an attacking host emits about $10^4 - 10^5$ packets. However, in real life, this value should be at least 10 times higher. This correction is introduced by the following two reasons. First, the simulations on the network simulator have shown the dependency on the routing topology. Figure 5 represents the graph of the expected number of packets in case of a binary tree topology and 3-ary tree topology compared to a linear topology. This figure shows that the number of packets required for path reconstruction may be significantly higher in case of a complicated topology even if only the attack traffic presents.

Second, the rate of attack packets emitted by a particular host can be quite low compared to the legitimate background Internet traffic. This reduces the probability that a router can sample an attack packet. Assuming an average rate of about 10^2 emitting attack packets per second per attacking host [4], we yield that the attacking hosts should generate traffic as long as 2-3 hours before a victim is able to reveal the whole path. The parts of the path, however, can be reconstructed after a few minutes, but the pieces of the path, obviously, do not disclose the origin of the attack, although they can provide valuable information pointing where to add traffic filters. In order to minimize the total number of attack packets required for path reconstruction, we have to raise the generating probability μ at least 10-20 times. However, this could be inappropriate due to the significant traffic overhead caused by the iTrace packets. Intention-driven iTrace can improve the situation and eliminate the influence of the topology and background traffic, although the propagation of the "intention" value through BGP updates may be quite slow, delaying the invocation of the inten-

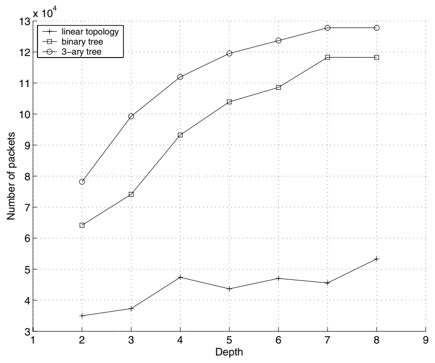


Fig. 5. The expected number of attacking packets per attacking host required for path reconstruction in iTrace in case of a linear and tree topology ($\mu = 1/2^{15}$)

tion driven mechanism. Another problem with intention driven iTrace is that it may cause instability in the routing mechanism due to frequent updates in the routing table.

3.3 Computation Overhead

PPM, as suggested by Savage and colleagues, suffers from enormous computation overhead in case of a large-scale DDOS. The complexity of reconstruction grows very rapidly and is upper bounded by $O(l \cdot n^8)$, where l is the maximal distance to the attackers and n is the number of attacking hosts. The large overhead is explained by the constraint to break the IP address of a router into eight pieces to fit each piece into the fragmentation field in the IP header. Consequently, the path reconstruction of all attacking sources can take hours or days [3].

The complexity of the path reconstruction in the traceback approach proposed by Song and Perrig depends primarily on the topology of the map of upstream routers instead of the number of attackers, and does not have a large overhead. The complexity of the path reconstruction is upper bounded by $O(l \cdot n)$ and therefore scales well, although the reconstruction procedure requires some intensive calculations [3]. The complexity of the reconstruction procedure in iTrace is upper bounded by $O(l \cdot n)$ and besides that, the reconstruction mechanism does not require significant and cumbersome calculations [5].

3.4 Robustness

In terms of robustness both the PPM approaches are insufficiently robust. The Savage algorithm gives a very high rate of false positives. Simulations in the network simulator have shown that false positives can even appear when the number of attackers is as low as five. Such behavior is explained, again, by a constraint to break the router's IP address into eight pieces, raising the probability of the accidentally incorrect reconstructed paths at the victim side. The Song and Perrig approach instead suffers from a possible high rate of false negatives. This is primary connected to the use of a map of upstream routers. If the map is not accurate or not correspond to the real topology, then the paths to some attackers cannot be reconstructed. In fact, no PPM approach is "collision-resistant" in the sense that ambiguous representation of the same edge increases the rate of false positives during the reconstruction procedure. The iTrace approach introduces a higher robustness, and does not suffer much from false positives. However, a host may not be discovered if it is sending a low volume of attack traffic. The robustness of iTrace is more influenced by insufficient information about the attack paths gathered at the victim, rather than ambiguous representation of information.

We would like to note that any traceback approach, which uses a path reconstruction mechanism, would suffer from false positives due to two main reasons. First, it is difficult to prove whether the path is reconstructed completely or partly. A partially reconstructed path is considered as a false negative, since the reconstruction procedure returns the IP address of a host, which is not actually involved in the attack and just lies somewhere in the middle of the path. Second, a single mistake on one step of the hop-by-hop path reconstruction procedure can imply multiple false negatives on the subsequent steps.

3.5 Deployment Overhead and Cost

One of the desired features of a traceback approach is incremental deployment into the current Internet structure, at low cost. The expenses for deployment of traceback should not exceed the losses from the denial of service activity. Unfortunately, all proposed approaches require a significantly high level of expenses since they require changes in routers present in the current Internet structure. The authors of the PPM scheme claim that their approach is incrementally deployable but this has not been thoroughly evaluated. For iTrace, the reconstruction of the whole path requires the participation of every other router on the path. Intention-driven iTrace has the drawback that it imposes a change in a somewhat sensitive BGP protocol.

4 A Modified iTrace Approach

As we have seen from the previous sections, the main drawbacks of the iTrace approach are a large number of attacking packets required for the path reconstruction and significant overhead caused by necessity to implement the iTrace mechanism on every router. While the first drawback can be (at least, theoretically) solved by tuning the generating probability μ, the second drawback requires a modification in the mechanism itself in order to add such a desirable characteristics as "incremental deployment" and reduce the overhead by implementing the iTrace mechanism only an a small amount of the routers.

An obvious improvement of the second drawback is the marking of the iTrace messages (issued by one router) on the other routers thereby creating a chain of the routers, which the message traverses on its path. However, this method supposes checking every packet in a router, which is inefficient. Instead, we suggest selecting packets with a relatively low probability (which we denote as a selecting probability η) from the queue in a router and if the selected packet is an iTrace message then mark it by adding some additional information to the body of the message. In other words, the router that implements iTrace should not only generate iTrace messages but also examine the randomly chosen packets and, if the examined packet is an iTrace message from another router, add its own IP address to a special field of the iTrace message. The reconstruction procedure at the victim side will use this information to fill in the "gaps" between the routers, which do not implement the iTrace mechanism.

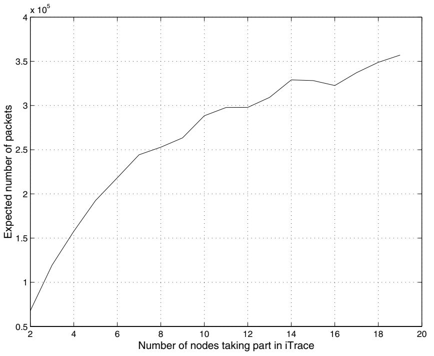


Fig. 6. The expected number of attacking packets required for path reconstruction in a modified iTrace approach in case of a linear topology ($\mu = 1/2^{15}$, $\eta = 1/2^{4}$)

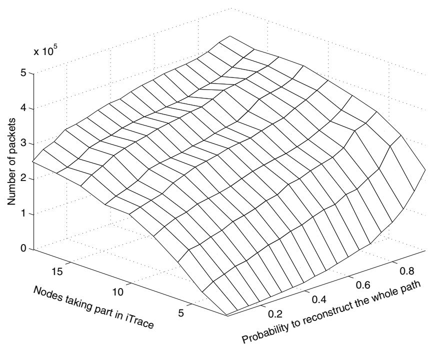


Fig. 7. The number of attacking packets required for reconstruction of the whole path with the given probability in case of a linear topology ($\mu = 1/2^{15}$, $\eta = 1/2^{4}$)

The selecting probability η should be low to avoid a processing overhead but, from the other hand, it should not be very low, since this decreases the probability of marking iTrace messages by the other routers. We suggest selecting the value for η from 0.05 to 0.1, because this gives an insignificant processing overhead on a router and, looking at the experiments, this value is sufficient for marking a reasonably high amount of iTrace messages.

We have proved that this approach allows successfully reconstruction of the path back to the router which is the nearest to the attaching host and which implements the modified iTrace mechanism. The reconstruction is possible even if only a part of the routers implements the modified iTrace mechanism. Moreover, a small amount of such routers, which located in some distance from each other, can do this work better than a large amount of routers disposed in a raw.

The following simulations were conducted in the network simulator. We have connected an attacking node, a "gap", a chain of the nodes, which implement the modified iTrace mechanism and a victim into a linear topology. The "gap" was represented by several nodes, which do not implement an iTrace mechanism at all. The length of the chain of the nodes, which implement the modified iTrace mechanism, varied from two up to twenty. The reconstruction procedure was extended to cope with the iTrace messages marked by the other nodes on its way. The purpose of these simulations was to figure out how the chain of nodes influences the path reconstruction.

The result of the simulations is shown in Figures 6 and 7. Figure 6 indicates the dependency of the expected number of attacking packets required for path reconstruction on the length of the chain. Figure 7 is a more extended analysis of our result. It shows the number of packets, which an attacking host should emit in order to reconstruct the whole path with a certain probability given in advance.

As the figures indicate, it is not desirable to have a long chain of routers which implements a modified iTrace approach because this complicates the reconstruction and requires more packets to complete the reconstruction. This phenomenon is explained as follows. As we mentioned above in Section 3.2, the reconstruction procedure makes a conclusion that the path is reconstructed completely if it receives no new samples of the path. The longer chain requires more packets for its reconstruction and emits more new samples of the path thereby delaying the reconstruction procedure with a conclusion that the reconstruction is complete.

5 Result

The desired characteristics of an IP traceback mechanism are that it requires a relatively small number of packets for path reconstructzion, low complexity of reconstruction, high robustness, and low deployment overhead and cost. All the parameters are equally important since a low evaluation of one of the parameters opposes using the method in practice.

Table 1 indicates that none of the proposed solutions satisfies all the desired parameters. Both PPM approaches have insufficient robustness in case of a DDOS attack but require fewer packets than iTrace for path reconstruction, whereas both of the iTrace approaches possess a relatively good robustness. The deployment overhead is high for all the methods (except the modified iTrace approach) since all of them assume some kind of alternation in routers in the Internet. The modified iTrace approach assumes alternation only in a small and carefully selected part of routers, reducing the deployment overhead. Summarizing the above, we infer that each of the suggested approaches is directed to solve one particular issue of the IP traceback problem but none can solve all of them.

Table 1. Summary of the evaluation of IP Traceback approaches

	packets	complexity	robustness	deployment overhead
Savage & colleagues	$4 \cdot 10^3$	$O(l \cdot n^8)$	*low*	*high*
Song & Perrig	$< 4 \cdot 10^3$	$O(l \cdot n)$	*medium*	*high*
ITrace	$> 10^6$	$O(l \cdot n)$	*high*	*high*
Intention driven	10^5	$O(l \cdot n)$	*high*	*very high*
Modified iTrace	$\approx 10^6$	$O(l \cdot n)$	*high*	*medium*

6 Summary

In this paper we have closely studied and evaluated four IP traceback approaches proposed during the last years. We have identified four parameters, which can be used for the comparison. These parameters are aimed at finding a time and cost efficient approach with high precision. Evaluation of the edge marking approach was based on previously published articles and on our own simulations, while the evaluation of the iTrace mechanism was based on our own research and simulations in the network simulator. We have also suggested a modified iTrace approach, which aims at reducing the deployment overhead.

7 Conclusions and Discussion

The detailed analysis has revealed two main disadvantages of the previously proposed approaches. First, the hop-by-hop path reconstruction is inefficient due to a significant computation overhead, or a long time spent for collecting the samples of the path. Second, the path reconstruction imposes changes in the core routing structure that is not profitable. With this in mind, a conclusion is that it may be a better approach to solve the IP traceback problem locally, concentrating on the first hop router, or the router that connects a local network to the rest of the Internet. The goal of all reconstruction algorithms is to find the sources of the attacking traffic, but the reconstruction of an attack path can actually only reveal the first router an attack packet has passed. Since the first router is of main interest, it would be desirable to find an algorithm that could reveal the identity of the first router, without requiring the participation of all the routers on the path. However, it would not be practical approach to rely only on the first-hop router because it might be compromised or damaged. The better solution would be to include a traceback support into the core Internet structure itself but to limit the overhead imposed by such changes. The modified iTrace approach that we have presented in this article is an example of such a solution. Our future research will focus on finding improvements in this approach.

Acknowledgement

This work has been supported by the project Radio Sphere Phase 1. We would also like to thank Olov Schelen for his valuable comments and help.

References

1. S. C. Lee and C. Shields, "Tracing the Source of Network Attack: A Technical, Legal and Societal Problem", Proc. of the 2001 IEEE Workshop on Information Assurance and Security, June 2001, United States Military Academy, West Point, NY, pp 239-246, `http://www.research.att.lists/ietfitrace/2000/09/msg00044.htm`
2. Stefan Savage, David Wetherall, Anna Karlin and Tom Anderson, "Practical Network Support for IP Traceback", Proc. of the ACM SIGCOMM conference, August 2000, Stockholm, Sweden, Computer Communication Review Vol. 30, No 4, October 2000, pp. 295-306,`http://www.cs.washington.edu/homes/savage/papers/Sigcomm00.ps`
3. Dawn X. Song and Adrian Perrig, "Advanced and Authenticated Marking Schemes for IP Traceback", Proc. of the IEEE infocom conference, april 2001, Anchorage, Alaska, `http://paris.cs.berkeley.edu/~perrig/projects/iptraceback/tr-iptrace.pdf`

4. David Moore, Geoffrey M. Voelker and Stefan Savage, "Inferring Internet Denial-of-Service Activity", Proc of the 10th USENIX Security Symposium, August 2001, Washington D.C., http://www.caida.org/outreach/papers/2001/BackScatter/usenixsecurity01.pdf
5. Steven M. Bellovin, "ICMP Traceback Messages", Internet Draft: draft-bellovin-itrace-00.txt, submitted Mar. 2000, expiration date Sep. 2000, http://www.research.att.com/~smb/papers/draft-bellovin-itrace-00.txt
6. S. Felix Wu, Lixia Zhang, Dan Massey, Allison Mankin, "Intention-Driven ICMP Traceback", Internet Draft: draft-wu-itrace-intention-00.txt, submission date Feb. 2001, expiration date Aug. 2001, http://www.ietf.org/internet-drafts/draft-ietf-itrace-intention-00.txt
7. Allison Mankin, Dan Massey, Chien-Long Wu, S. Felix Wu, Lixia Zhang, "On Design and Evaluation of Intention-Driven ICMP Traceback", Proc. of 10th IEEE International Conference on Computer Communications and Networks, Oct.2001, Scotsdale Arizona, http://irl.cs.ucla.edu/papers/Intention-iTrace.pdf
8. Alex C. Snoeren et al., "Hash-Based IP Traceback", Proc. of the ACM SIGCOMM conference •2001, San Diego, CA, Computer Communication Review Vol. 31, No 4, October 2001, pp. 3-14, http://nms.lcs.mit.edu/~snoeren/papers/spie-sigcomm.pdf
9. Drew Dean, Matt Franklin, Adam Stubblefield, "An Algebraic Approach to IP Traceback", Xerox PARC, Rice University, August 2000, http://www.cs.rice.edu/~astubble/algtrace.pdf
10. The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns
11. Vern Paxson, "An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks", ACM SIGCOMM Computer Communication Review Vol. 31, No 3, July 2001, http://www.acm.org/sigcomm/ccr/archive/2001/jul01/ccr-200107-paxson.pdf
12. Robert Stone "CenterTrack: An IP Overlay Network for Tracking DoS Floods", Proceedings of the 9th USENIX Security Symposium, Aug 2000, http://www.usenix.org/publications/library/proceedings/sec2000/full_{p}apers/stone/stone.pdf
13. P. Fergusson and D.Seine, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing",RFC2827, May 2000, http://www.ietf.org/rfc/rfc2827.txt

Security against Inference Attacks on Negative Information in Object-Oriented Databases

Yasunori Ishihara, Shuichiro Ako, and Toru Fujiwara

Graduate School of Information Science and Technology
Osaka University
Toyonaka, Osaka, 560-8531 Japan

Abstract. Inference attacks mean that a user derives information on the execution results of unauthorized queries from the execution results of authorized queries. Although many studies so far focus on only inference of positive information (i.e., which object is the execution result of a given unauthorized query), negative information (i.e., which object is never the execution result of a given unauthorized query) is also sensitive. In this paper, we define the following two types of security problems against inference attacks on given negative information: (1) Is the information secure under a given database instance? (2) Is it secure under any database instance of a given database schema? It is shown that the first problem is decidable in polynomial time in the description size of the database instance while the second one is undecidable. A decidable sufficient condition for given negative information to be secure under any database instance of a given database schema is also proposed.

1 Introduction

In recent years, various authorization models for object-oriented databases (OODBs) have been proposed and studied. Among them, the method-based authorization model [7,13] is one of the most elegant models since it is in harmony with the concept that "an object can be accessed only via its methods" in the object-oriented paradigm. In the model, an authorization A for a user u can be represented as a set of rights $m(c_1, \ldots, c_n)$, which means that u can *directly* invoke method m on any tuple $(o_1, \ldots, o_n)$ of objects such that o_i is an object of class c_i with $1 \leq i \leq n$. On the other hand, even if $m(c_1, \ldots, c_n) \notin A$, u can invoke m *indirectly* through another method execution in several models, e.g., protection mode in [2]. Although such indirect invocations are useful for data hiding [2], they may also allow *inference attacks* in some situations.

Example 1. Consider the following database schema: Employee, Host, and Room are classes representing employees, hosts, and rooms, respectively. Method computer returns the host which a given employee uses, method location returns the room in which a given host is placed, and method office, which returns the room occupied by a given employee, is implemented as office(x) = location(computer(x)).

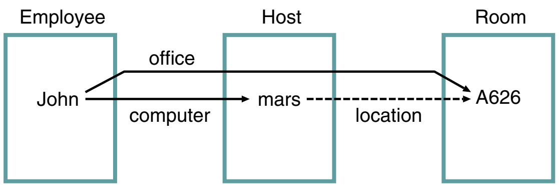


Fig. 1. Inference of positive information.

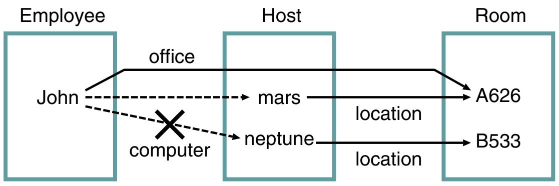


Fig. 2. Inference of negative information.

Now suppose that the physical computer network is top-secret information. In this case, an authorization for a user u may be the one shown in Figure 1, where a solid (resp. dotted) arrow denotes an authorized (resp. unauthorized) method to u. Assume that u have obtained that $\mathsf{computer}(\mathsf{John}) = \mathsf{mars}$ and $\mathsf{office}(\mathsf{John}) = \mathsf{A626}$ using the authorized methods. Also assume that u knows the implementation body of office as its behavioral specification. Then, u can infer that $\mathsf{location}(\mathsf{mars}) = \mathsf{A626}$, although executing $\mathsf{location}(\mathsf{mars})$ is prohibited.

In the above example, the positive information "$\mathsf{location}(\mathsf{mars}) = \mathsf{A626}$" is not secure against inference attacks. It is important for database administrators to know whether top-secret information is secure against inference attacks.

In [8,9], two security problems against inference attacks on OODBs are discussed. These articles focus on inference of only positive information, i.e., inference of the execution result of a given unauthorized query (see Example 1). However, as illustrated below, negative information can also be inferred.

Example 2. Consider again the schema in Example 1. In this case, suppose that the relationship between employees and hosts is top-secret information and therefore the authorization of u is set as shown in Figure 2. Then, u knows that $\mathsf{location}(\mathsf{neptune}) = \mathsf{B533}$ and $\mathsf{office}(\mathsf{John}) = \mathsf{location}(\mathsf{computer}(\mathsf{John})) = \mathsf{A626}$, similarly to Example 1. Since $\mathsf{A626}$ and $\mathsf{B533}$ are different rooms, u can conclude that $\mathsf{computer}(\mathsf{John}) \neq \mathsf{neptune}$.

In this paper, inference of negative information is formalized under a model of OODBs. Under the formalization, we consider the following security problems:

1. *The instance security problem (ISP)*: Under a given database instance, is it impossible to infer an object that is never the execution result of a given unauthorized query?

2. *The schema security problem (SSP)*: Under any database instance of a given database schema, is it impossible to infer an object that is never the execution result of a given unauthorized query?

It is shown that the ISP is decidable in polynomial time in the description size of the database instance while the SSP is undecidable. We propose a decidable sufficient condition for given negative information to be secure under any database instance of a given database schema.

In this paper we discuss "logical" inference in OODBs in the sense that the result of the inference is always true. The inference in statistical databases [4] is a kind of logical inference. Ref. [6] proposes a mechanism that accomplishes maximum data availability as long as given sensitive information is secure against logical inference. Ref. [14] focuses on logical inference in OODBs. Besides inferability of the result of a method execution, the article introduces the notion of controllability, which means that a user can control (alter arbitrarily) an attribute-value of an object in a database instance. We do not consider controllability since our query language does not support update operations for database instances. However, since our query language supports recursion while the one in [14] does not, detecting inferability in our formalization is not trivial.

On the other hand, many of the researches concentrate on "statistical" inference, i.e., inference under some statistical assumptions. Ref. [3] discusses the inference based on Bayesian methods. In [15], a quantitative measure of inference risk is formally defined. In [10], the security against statistical inference is defined based on information theory.

The rest of this paper is organized as follows. Section 2 defines OODBs. Section 3 defines attacker's inference and the security problems. Sections 4 and 5 discuss the two security problems. Section 6 sums up the paper. Because of the space limitation, we rarely provide examples in this paper. Refer to [8] for some explanatory examples of the definitions of OODBs and inference attacks.

2 A Formal Model of OODBs

We adopt *method schemas* [1] as a formal model of OODBs. Method schemas have such basic features of OODBs as method overloading, dynamic binding, and complex objects. The semantics is simply defined based on term rewriting. In this section, we first introduce some notations and concepts for term rewriting. Then, we restate the original definition of method schemas.

2.1 Notations

Let F be a family of disjoint sets F_0, F_1, F_2,..., where, for a nonnegative integer n, F_n is a set of function symbols of arity n. For a countable set X of variables, let $T_F(X)$ denote the set of all the terms freely generated by F and X. For a set V, let V^n denote the Cartesian product $\underbrace{V \times \cdots \times V}_{n}$. For a term $t \in T_F(X)$, an n-tuple $\mathbf{t} = (t_1, \ldots, t_n) \in (T_F(X))^n$ of terms, and an n-tuple $\mathbf{x} = (x_1, \ldots, x_n) \in$

X^n of variables, let $t[\mathbf{t}/\mathbf{x}]$ denote the term obtained by simultaneously replacing every x_i in t with t_i for $1 \leq i \leq n$. Hereafter, we often use a bold letter $\mathbf{v}$ to mean $(v_1, \ldots, v_n)$ without explicitly defining it when n is irrelevant or obvious from the context. Also, we write $v \in \mathbf{v}$ if $v = v_i$ for some i.

An *occurrence* of a term t is a sequence of positive integers representing (the position of) a subterm of t. For example, $1 \cdot 2$ of $f(f(x, g(x)), g(x)))$ specifies the first $g(x)$, i.e., the second argument of the first argument of the outermost f. Let $R(t)$ denote the set of occurrences of t. The subterm of t at occurrence r is denoted t/r. The replacement in t of t' at occurrence r, denoted $t[r \leftarrow t']$, is defined as follows:

- $t[\varepsilon \leftarrow t'] = t'$, where ε denotes the empty sequence;
- $f(t_1, \ldots, t_i, \ldots, t_n)[i \cdot r \leftarrow t'] = f(t_1, \ldots, t_{i-1}, t_i[r \leftarrow t'], t_{i+1}, \ldots, t_n)$.

2.2 Database Schemas

Let C be a finite set of *class names* (or simply classes). Let M be a family of mutually disjoint finite sets M_0, M_1, M_2,..., where, for a nonnegative integer n, M_n is a set of function symbols (or often called *method names*) of arity n. Each M_n is partitioned into $M_{\mathrm{b},n}$ and $M_{\mathrm{c},n}$. Let $M_\mathrm{b} = \bigcup_{n \geq 0} M_{\mathrm{b},n}$ and $M_\mathrm{c} = \bigcup_{n \geq 0} M_{\mathrm{c},n}$. Each $m_\mathrm{b} \in M_\mathrm{b}$ (resp. $m_\mathrm{c} \in M_\mathrm{c}$) is called a *base method name* (resp. *composite method name*). We say that M is a *method signature*.

Definition 1. *Let* $\mathbf{c} \in C^n$. *A* base method definition *of* $m_\mathrm{b} \in M_{\mathrm{b},n}$ *at* $\mathbf{c}$ *is a pair* $(m_\mathrm{b}(\mathbf{c}), c)$ *for some* $c \in C$. *A* composite method definition *of* $m_\mathrm{c} \in M_{\mathrm{c},n}$ *at* $\mathbf{c}$ *is a pair* $(m_\mathrm{c}(\mathbf{c}), t)$ *for some* $t \in T_M(\{x_1, \ldots, x_n\})$.

For $1 \leq i \leq n$, let o_i be an object of class c_i (see Def. 4 for the formal definition of objects). Informally, the above base method definition declares that the application of m_b to $\mathbf{o} = (o_1, \ldots, o_n)$ results in an object of c or its subclass, while the above composite method definition states that the application of m_c to $\mathbf{o}$ results in term rewriting starting from $t[\mathbf{o}/\mathbf{x}]$.

Definition 2. *A* method schema *[1] S is a 5-tuple* $(C, \leq, M, \Sigma_\mathrm{b}, \Sigma_\mathrm{c})$*, where:*

1. *C is a finite set of class names,*
2. *$\leq$ is a partial order on C representing a class hierarchy,*
3. *M is a method signature,*
4. *Σ_b is a set of base method definitions, and*
5. *Σ_c is a set of composite method definitions.*

For every combination $\mathbf{c} \in C^n$ and $m \in M_n$, there must exist at most one method definition of m at $\mathbf{c}$.

When $c' \leq c$, we say that c' is a subclass of c and c is a superclass of c'. We naturally extend $\leq$ to n-tuples of classes as follows: For two tuples $\mathbf{c} = (c_1, \ldots, c_n)$ and $\mathbf{c}' = (c'_1, \ldots, c'_n)$, we write $\mathbf{c} \leq \mathbf{c}'$ *iff* $c_i \leq c'_i$ for all i.

Define the size $\|t\|$ of a term t as $|R(t)|$, where $|X|$ denotes the number of elements of a set X. That is, $\|t\|$ is the number of occurrences of t.

2.3 Inheritance

Let $\mathbf{c} \in C^n$ and $m \in M_n$. By Def. 2, the method definition of m at $\mathbf{c}$ may not exist. In this case, the definition of m at the smallest superclass of $\mathbf{c}$ is "inherited" by $\mathbf{c}$. The inherited method definition is called *resolution* and defined as follows:

Definition 3. *Let $S = (C, \leq, M, \Sigma_{\mathrm{b}}, \Sigma_{\mathrm{c}})$, $m_{\mathrm{b}} \in M_{\mathrm{b},n}$, and $\mathbf{c} \in C^n$. Suppose that $(m_{\mathrm{b}}(\mathbf{c}'), c') \in \Sigma_{\mathrm{b}}$ is the base method definition of m_{b} at the smallest $\mathbf{c}'$ above $\mathbf{c}$, i.e., whenever $(m_{\mathrm{b}}(\mathbf{c}''), c'') \in \Sigma_{\mathrm{b}}$ and $\mathbf{c} \leq \mathbf{c}''$, it is the case that $\mathbf{c}' \leq \mathbf{c}''$. The* resolution *$Res(m_{\mathrm{b}}(\mathbf{c}))$ of m_{b} at $\mathbf{c}$ is defined as c'. If such a unique base method definition does not exist, then $Res(m_{\mathrm{b}}(\mathbf{c}))$ is* undefined*, denoted $\perp$. The resolution of a composite method is defined in the same way.*

2.4 Database Instance

A database instance of a method schema assigns a set of objects to each class name. Also, it gives the semantics of base methods.

Definition 4. *A* database instance *of a method schema S is a pair $I = (\nu, \mu)$ with the following properties:*

1. *To each $c \in C$, ν assigns a finite disjoint set $\nu(c)$ of* object identifiers *(or simply,* objects*). Each $o \in \nu(c)$ is called an object of class c. Let $O_I = \bigcup_{c \in C} \nu(c)$. For $\mathbf{c} = (c_1, \ldots, c_n)$, let $\nu(\mathbf{c})$ denote $\nu(c_1) \times \cdots \times \nu(c_n)$.*
2. *For each $m_{\mathrm{b}} \in M_{\mathrm{b},n}$, $\mu(m_{\mathrm{b}})$ is a mapping from O_I^n to O_I which satisfies the following two conditions. Let $\mathbf{c}$, $\mathbf{c}' \in C^n$.*
   (a) If $Res(m_{\mathrm{b}}(\mathbf{c})) = c'$, then $\mu(m_{\mathrm{b}}) |_{\nu(\mathbf{c})}$ is a partial mapping to $\bigcup_{c \leq c'} \nu(c)$, where "$|$" denotes that the domain of $\mu(m_{\mathrm{b}})$ is restricted to $\nu(\mathbf{c})$.
   (b) If $Res(m_{\mathrm{b}}(\mathbf{c})) = \perp$, then $\mu(m_{\mathrm{b}})$ is undefined everywhere in $\nu(\mathbf{c})$.
   If $\mu(m)(\mathbf{o})$ is undefined, then we write $\mu(m)(\mathbf{o}) = \perp$.

The description size of I is $O(|O_I|^k)$, where k is the maximum arity of the methods in the schema.

2.5 Method Execution

A term in $T_M(O_I)$ is called an *instantiated term*. The *one-step execution relation* $\rightarrow_I$ on $T_M(O_I)$ is defined based on the leftmost innermost reduction strategy.

Definition 5. *For a term $t \in T_M(O_I)$, let $m(\mathbf{o})$ ($\mathbf{o} \in \nu(\mathbf{c})$) be the leftmost subterm of t at occurrence r.*

1. *If $m \in M_{\mathrm{b}}$ and $\mu(m)(\mathbf{o}) \neq \perp$, then $t \rightarrow_I t[r \leftarrow \mu(m)(\mathbf{o})]$.*
2. *If $m \in M_{\mathrm{c}}$ and $Res(m(\mathbf{c})) = t' \neq \perp$, then $t \rightarrow_I t[r \leftarrow t'[\mathbf{o}/\mathbf{x}]]$.*

Note that, by Def. 5, for any instantiated term t, there exists at most one term t' such that $t \to_I t'$. That is, every execution is deterministic. Taking the leftmost reduction strategy is just for ensuring that every execution is deterministic. On the other hand, taking the innermost reduction strategy (i.e., rewriting only the term in the form of $m(\mathbf{o})$) is essential since the definition of m cannot be bound before knowing the classes of the arguments of m.

Let $\to_I^*$ be the reflexive and transitive closure of $\to_I$. The *execution result* of t, denoted $t{\downarrow}_I$, is a term t' such that $t \to_I^* t'$ and there exists no t'' such that $t' \to_I t''$. If $t{\downarrow}_I \in O_I$, then the execution of t is *successful*, and if $t{\downarrow}_I \notin O_I$, then the execution of t is *aborted*. In both cases (i.e., if $t{\downarrow}_I$ exists), the execution of t is *terminating*. On the other hand, if $t{\downarrow}_I$ does not exist, then the execution of t is *nonterminating*.

3 Inference Attacks

3.1 Definition of Inference

We adopt the following simple but general method-based authorization model because discussing authorization models is not our main purpose.

Definition 6. *Let $S = (C, \leq, M, \Sigma_{\rm b}, \Sigma_{\rm c})$. A* right *is a term in the form of $m(\mathbf{c})$, where $m \in M_n$ and $\mathbf{c} \in C^n$. An* authorization *A is a finite set of rights.*

Intuitively, $m(\mathbf{c}) \in A$ means that the corresponding user is authorized to directly invoke method m on any tuple $\mathbf{o}$ of objects such that $\mathbf{o} \in \nu(\mathbf{c})$.

For theoretical convenience, we introduce a special object $o_\perp \notin O_I$ denoting aborted execution. Extend $\to_I$ to the relation over $T_M(O_I \cup \{o_\perp\})$ as follows:

1. If the execution of t under I is aborted, then $t \to_I o_\perp$.
2. If $o_\perp \in \mathbf{o}$, then $m(\mathbf{o}) \to_I o_\perp$.

Define $t{\downarrow}_I = o_\perp$ if $t \to_I^* o_\perp$. Thus, $t{\downarrow}_I$ is defined whenever the execution of t is terminating. In the rest of this paper, $\to_I$ denotes this extended rewrite relation.

In this paper, the information obtained from the database is represented by a predicate $\mathrm{EQ}(t, t')$. Here, $\mathrm{EQ}(t, t')$ (resp. $\neg\mathrm{EQ}(t, t')$) means that the user knows that the execution results of t and t' are the same object (resp. different objects). For example, user u obtains $\mathrm{EQ}(\mathsf{location}(\mathsf{mars}), \mathsf{A626})$ in Example 1 and $\neg\mathrm{EQ}(\mathsf{computer}(\mathsf{John}), \mathsf{mars})$ in Example 2.

As stated in Examples 1 and 2, we assume that user u can obtain the following three kinds of information (i.e., equalities and non-equalities) directly from a database instance $I = (\nu, \mu)$.

(∗1) User u knows the execution result of $m(\mathbf{o})$ if the execution is terminating. Formally, if $m(\mathbf{c}) \in A$, $\mathbf{o} \in \nu(\mathbf{c})$, and $m(\mathbf{o}){\downarrow}_I$ exists, then u obtains $\mathrm{EQ}(m(\mathbf{o}), m(\mathbf{o}){\downarrow}_I)$.

(∗2) User u knows the behavioral specification of a composite method $m_{\rm c}$ at $\mathbf{c}$ if executing $m_{\rm c}(\mathbf{o})$ ($\mathbf{o} \in \nu(\mathbf{c})$) is authorized. Formally, if $m_{\rm c}(\mathbf{c}) \in A$, $Res(m_{\rm c}(\mathbf{c})) = t$, and $\mathbf{o} \in \nu(\mathbf{c})$, then u obtains $\mathrm{EQ}(m_{\rm c}(\mathbf{o}), t[\mathbf{o}/\mathbf{x}])$.

(∗3) User u can distinguish different objects. Formally, for every pair of distinct objects o and o' in $O_I \cup \{o_\perp\}$, u obtains $\neg\mathrm{EQ}(o, o')$.

A behavioral specification is usually more abstract and has less information than its implementation (i.e., $Res(m(\mathbf{c}))$). Our formulation is conservative in the sense that if a term is secure in our model, then it will be secure in the real world.

Now, let $K^+_{I,A}$ denote the set of EQ's by (∗1) and (∗2), and let $K^-_{I,A}$ be the set of EQ's by (∗3). Let AX be the set of the following Horn clauses:

(AX1) $\mathrm{EQ}(x, x)$,
(AX2) $\neg\mathrm{EQ}(x, y) \vee \mathrm{EQ}(y, x)$,
(AX3) $\neg\mathrm{EQ}(x, y) \vee \neg\mathrm{EQ}(y, z) \vee \mathrm{EQ}(x, z)$,
(AX4) $\neg\mathrm{EQ}(x, y) \vee \mathrm{EQ}(m(x_1, \ldots, x, \ldots, x_n), m(x_1, \ldots, y, \ldots, x_n))$,
(AX5) $\mathrm{EQ}(m(x_1, \ldots, o_\perp, \ldots, x_n), o_\perp)$,

where (AX1)–(AX4) are axioms of equality and (AX5) represents the property of $o_\perp$. Define $K_{I,A} = K^+_{I,A} \cup K^-_{I,A} \cup AX$. The information that can be inferred by the user is defined as the set of all the logical consequences of $K_{I,A}$, denoted $\{f \mid K_{I,A} \models f\}$. Especially, the negative information that can be inferred by the user is $\{\neg\mathrm{EQ}(t, o) \mid K_{I,A} \models \neg\mathrm{EQ}(t, o), o \in O_I\}$.

The above definition of inference is reasonable under the next assumption:

Assumption 1. *User u knows neither what O_I is nor what C is.*

If u knows O_I, then u can conclude $\mathrm{EQ}(t, o)$ by obtaining $\neg\mathrm{EQ}(t, o')$ for every $o' \in O_I - \{o\}$. Such inference is not captured by the axioms in AX. If u knows C, the equalities obtained by (∗2) may not be ground (i.e., may include some variables). See the discussion in [8,9] for detail. In many cases, this assumption can be satisfied by just hiding O_I and C from u.

3.2 Two Security Problems

We formalize the two security problems against inference of negative information. The first one, the *instance security problem (ISP)*, is to determine whether for a given database instance I, a query $\tau \in T_M(O_I)$, and an authorization A, there is no $o \in O_I$ such that $K_{I,A} \models \neg\mathrm{EQ}(\tau, o)$. The second one, the *schema security problem (SSP)*, is to determine whether for a given database schema S, a query $\tau \in T_M(C)$, and an authorization A, there is no database instance $I = (\nu, \mu)$ of S such that $K_{I,A} \models \neg\mathrm{EQ}(\tau[\mathbf{o}/\mathbf{c}], o)$ for some $o \in O_I$ and $\mathbf{o} \in \nu(\mathbf{c})$. In Section 4, we show that the ISP is decidable in polynomial time in the description size of the database instance. In Section 5, we show that the SSP is undecidable, and propose a decidable sufficient condition for a given schema to be secure.

4 The Instance Security Problem

In this section, we show that the ISP is decidable. To do this, we use the notion of congruence closure defined below.

Definition 7. *Let B be a binary relation on $T_M(O_I \cup \{o_\perp\})$. The* congruence closure *of B, denoted B^*, is the least (finest) equivalence relation such that*

- *B^* includes B; and*
- *B^* satisfies substitutivity, i.e., if $(t, t') \in B^*$, then $(m(t_1, \ldots, t, \ldots, t_n), m(t_1, \ldots, t', \ldots, t_n)) \in B^*$ for any $m \in M$.*

Let $B_{I,A} = \{(t, t') \mid \mathrm{EQ}(t, t') \in K^+_{I,A}\}$. Also, for $\tau \in T_M(O_I)$, let $B_{I,A}(\tau, o)$ denote $B_{I,A} \cup \{(\tau, o)\}$. For given I and A, $B_{I,A}$ can be computed in polynomial time in the description size of the database instance [9]. Since every term in $B_{I,A}$ (and also $B_{I,A}(\tau, o)$) is ground (i.e., includes no variables), the congruence closure can be computed in polynomial time in the description size of the database instance, by using the fast algorithms proposed in [5].

The axioms (AX1)–(AX4) are in the form of Horn clauses, not in the ordinary form such as "if $\mathrm{EQ}(x, y)$ then $\mathrm{EQ}(y, x)$," etc. However, as is the ordinary case, the following well-known property holds:

Lemma 1. *Let $t \in T_M(O_I)$ and $o \in O_I$. $(K^+_{I,A} \cup AX) \models EQ(t, o)$* iff *$(t, o) \in B^*_{I,A}$.*

Proof. By Proposition 1.5 in [11], a positive literal $\mathrm{EQ}(t, o)$ is a logical consequence *iff* there is a finite number of hyperresolution steps for the Horn clauses in AX. Here, each hyperresolution step is equivalent to an application of an ordinary axiom of equality such as "if $\mathrm{EQ}(x, y)$ then $\mathrm{EQ}(y, x)$." □

The following lemma states that disjunction of positive EQ's is decomposable into some single EQ's.

Lemma 2. *Let K be a set of Horn clauses including only $EQ(\cdots)$ as predicates. If $K \models \bigvee_{i=1}^{n} EQ(t_i, t'_i)$, then $K \models EQ(t_i, t'_i)$ for some $1 \le i \le n$.*

Proof. Since K is a set of Horn clauses, K has the minimum model H among its Herbrand models. Every model is characterized by a set of EQ's, and hence, $\mathrm{EQ}(t_i, t'_i) \in H$ for some $1 \le i \le n$. By the minimality of H, $K \models \mathrm{EQ}(t_i, t'_i)$. □

Theorem 1. *Let $t \in T_M(O_I)$ and $o \in O_I$. $K_{I,A} \models \neg EQ(t, o)$* iff *there exist $o_1, o_2 \in O_I \cup \{o_\perp\}$ such that $o_1 \not\equiv o_2$ and $(o_1, o_2) \in (B_{I,A}(t, o))^*$.*

Proof. For the *only if* part, suppose that $K_{I,A} \models \neg\mathrm{EQ}(t, o)$. Then, we have $(K^+_{I,A} \cup AX \cup \{\mathrm{EQ}(t, o)\}) \models \bigvee\{\mathrm{EQ}(o_1, o_2) \mid \neg\mathrm{EQ}(o_1, o_2) \in K^-_{I,A}\}$. By Lemma 2, there exists $\mathrm{EQ}(o_1, o_2)$ such that $(K^+_{I,A} \cup AX \cup \{\mathrm{EQ}(t, o)\}) \models \mathrm{EQ}(o_1, o_2)$. By (∗3), $o_1 \not\equiv o_2$. Also, by Lemma 1, $(o_1, o_2) \in (B_{I,A}(t, o))^*$.

As for the *if* part, $(K^+_{I,A} \cup AX \cup \{\mathrm{EQ}(t, o)\}) \models \mathrm{EQ}(o_1, o_2)$ by (∗3) and Lemma 1. Since $\neg\mathrm{EQ}(o_1, o_2) \in K^-_{I,A}$, we have $(K_{I,A} \cup \{\mathrm{EQ}(t, o)\}) \models \text{false}$. Thus $K_{I,A} \models \neg\mathrm{EQ}(t, o)$. □

Theorem 1 induces a straightforward algorithm for the ISP. That is, check whether $(o_1, o_2) \in (B_{I,A}(t, o))^*$ for each triple $o, o_1, o_2 \in O_I \cup \{o_\perp\}$ with $o \not\equiv o_\perp$ and $o_1 \not\equiv o_2$. This algorithm takes polynomial time in the description size of I.

5 The Schema Security Problem

5.1 Undecidability

First, we show the undecidability of the SSP.

Theorem 2. *The schema security problem is undecidable for schemas with binary methods.*

Proof. In [8], the "positive version" of the SSP for schemas with binary methods is shown to be undecidable by reducing the Post's Correspondence Problem (PCP) to the positive version. In the reduction, each database instance I is regarded as a candidate for a solution to a given PCP instance. If I is actually a solution, then execution of a term, say $m(o)$, is successful under I. Otherwise, $m(o)$ is nonterminating. By slightly modifying the reduction in [8], we can construct a schema with the following properties:

- If I is a solution, then the execution of a term, say $m'(o)$, is aborted under I. Otherwise, the execution of $m'(o)$ is successful.
- There are methods m_1 and m_2 such that for every c_1, $c_2 \in C$, $Res(m_1(c_1, c_2)) = x_2$ and $Res(m_2(c_1)) = m_1(m'(x_1), x_1)$.

Let c be the class to which o belongs. Let $A = \{m_1(c, c), m_2(c)\}$ and $\tau = m'(c)$. The execution of $m_2(o)$ is successful *iff* that of $\tau[o/c] = m'(o)$ is successful.

Suppose that the PCP instance has a solution I. Then, from the execution result of $m_2(o)$ under I, the user obtains that $m'(o)$ is aborted (i.e., $K_{I,A} \models \neg\mathrm{EQ}(m'(o), o')$ for every $o' \in \nu(c)$) using $(*1)$–$(*3)$ and (AX3)–(AX4). On the other hand, suppose that the PCP instance has no solution. Let I be an arbitrary database instance. In this case, all the equalities relevant to m' that the user can obtain are only $\mathrm{EQ}(m_1(m'(o), o), o)$, $\mathrm{EQ}(m_1(o, o), o)$, and $\mathrm{EQ}(m_2(o), o)$ by $(*1)$ and $(*2)$, and therefore, $K_{I,A} \cup \{\mathrm{EQ}(m'(o), o')\}$ is consistent for every $o' \in O_I$. Thus, $K_{I,A} \not\models \neg\mathrm{EQ}(m'(o), o')$ for every $o' \in O_I$. □

5.2 A Sufficient Condition for the Security

In order to explain the proposed sufficient condition for the security, we use another definition of inference. In [8], inference is defined through a rewriting relation $\Rightarrow_{I,A}$ on $T_M(O_I \cup \{o_\perp\})$. The formal definition of $\Rightarrow_{I,A}$ is omitted because of the space limitation.

Let $t \in T_M(O_I)$ and $o \in O_I$. Let $\Rightarrow^*_{I,A}$ denote the reflexive and transitive closure of $\Rightarrow_{I,A}$. By Lemma 1 and [8], inference of positive information $\mathrm{EQ}(t, o)$ is characterized as follows: $(K^+_{I,A} \cup AX) \models \mathrm{EQ}(t, o)$ *iff* $(t, o) \in B^*_{I,A}$ *iff* $t \Rightarrow^*_{I,A} o$. The next lemma states that inference of negative information $\neg\mathrm{EQ}(t, o)$ can also be characterized by $\Rightarrow^*_{I,A}$.

Lemma 3. *Let $\tau \in T_M(O_I)$ and $o \in O_I$. The followings are equivalent:*

1. *$K_{I,A} \models \neg EQ(\tau, o)$.*
2. *There exist $o_1, o_2 \in O_I \cup \{o_\perp\}$ and $t \in T_M(O_I)$ such that $o_1 \not\equiv o_2$, $\tau = t/r$, $t \Rightarrow^*_{I,A} o_1$, and $t[r \leftarrow o] \Rightarrow^*_{I,A} o_2$.*

Proof. (1⇒2) Assume 1. By Theorem 1, there exist some $o_1, o_2 \in O_I \cup \{o_\perp\}$ such that $o_1 \neq o_2$ and $(o_1, o_2) \in (B_{I,A}(\tau, o))^*$. Since $(o_1, o_2) \in (B_{I,A}(\tau, o))^*$ but $(o_1, o_2) \notin B^*_{I,A}$, there must be some $t \in T_M(O_I)$ such that $\tau = t/r$ for some occurrence r, $(t, o_1) \in B^*_{I,A}$, and $(t[r \leftarrow o], o_2) \in B^*_{I,A}$. By the property of $\Rightarrow^*_{I,A}$, we have 2.

(1⇐2) Assume 2. By the property of $\Rightarrow^*_{I,A}$, $(t, o_1) \in B^*_{I,A}$ and $(t[r \leftarrow o], o_2) \in B^*_{I,A}$. Therefore, $(o_1, o_2) \in (B_{I,A}(\tau, o))^*$ and we have 1 by Theorem 1. □

Similarly to [8], the main idea of the sufficient condition is to introduce "class-level" rewriting relation which "conservatively" approximate $\Rightarrow_{I,A}$, i.e., if $\tau \in T_M(C)$ is insecure, then τ is reducible to a class c by the class-level rewriting relation. To do this, we introduce a special class $c_\perp$ as the class of $o_\perp$, and then, we extend ν so that $\nu(c_\perp) = \{o_\perp\}$. Intuitively, each $t \in T_M(C \cup \{c_\perp\})$ is considered as the set of instantiated terms $\{t[\mathbf{o}/\mathbf{c}] \mid \mathbf{o} \in \nu(\mathbf{c})\}$. The "execution result" $E(t)$ of t is defined as follows: $c \in E(t)$ *iff* there is an instance $I = (\nu, \mu)$ such that $t[\mathbf{o}/\mathbf{c}]\downarrow_I \in \nu(c)$ for some $\mathbf{o} \in \nu(\mathbf{c})$. Unfortunately, we cannot compute E exactly in general [1]. However, we can compute $Z : T_M(C \cup \{c_\perp\}) \to 2^{C \cup \{c_\perp\}}$ such that $Z(t) \supseteq E(t)$ for every t [12]. We use such Z to approximate $\Rightarrow_{I,A}$.

Definition 8. *Define $P_{S,A,Z}$ as the minimum set of rewriting rules $\rhd_{S,A,Z}$ on $T_M(C \cup \{c_\perp\})$ satisfying the following three conditions:*

(A) If $m(\mathbf{c}) \in A$, then $P_{S,A,Z}$ contains $m(\mathbf{c}) \rhd_{S,A,Z} c$ for each $c \in Z(m(\mathbf{c}))$.
(B) If $m_c(\mathbf{c}) \in A$, $m_c \in M_{c,n}$, and $Res(m_c(\mathbf{c})) = t \neq \perp$, then $P_{S,A,Z}$ contains $t[\mathbf{c}/\mathbf{x}] \rhd_{S,A,Z} c$ for each $c \in Z(t[\mathbf{c}/\mathbf{x}])$.
(C) If $P_{S,A,Z}$ contains $t \rhd_{S,A,Z} c$ and $t'' \rhd_{S,A,Z} c''$ such that t'' is a proper subterm of t at r'', then $P_{S,A,Z}$ contains $t[r'' \leftarrow c''] \rhd_{S,A,Z} c'$ for each $c' \in Z(t[r'' \leftarrow c''])$.

Define $\Rightarrow_{S,A,Z}$ as the one-step reduction relation by $\rhd_{S,A,Z}$. Let $\Rightarrow^*_{S,A,Z}$ denote the reflexive and transitive closure of $\Rightarrow_{S,A,Z}$. Note that for a given $t \in T_M(C)$, it is decidable whether $t \Rightarrow_{S,A,Z} c$ for some $c \in C \cup \{c_\perp\}$, because in every rule in $P_{S,A,Z}$ the size of the right-hand side is less than that of the left-hand side.

Lemma 4. *Let $t \in T_M(C)$ and $c \in C$. Suppose that there is an instance $I = (\nu, \mu)$ such that $t[\mathbf{o}/\mathbf{c}] \Rightarrow^*_{I,A} o$ for some $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in \nu(c)$. Then, $t \Rightarrow^*_{S,A,Z} c$.*

Proof. Immediate from Theorem 2 in [8]. □

We have the following theorem from Lemmas 3 and 4.

Theorem 3. *Let $\tau \in T_M(C)$ and $c \in C$. Suppose that for every term $t \in T_M(C)$ such that $\tau = t/r$ for some occurrence r of t, at least one of the following conditions holds:*

– *There is no $c_1 \in C \cup \{c_\perp\}$ such that $t \Rightarrow^*_{S,A,Z} c_1$.*
– *There is no $c_2 \in C \cup \{c_\perp\}$ such that $t[r \leftarrow c] \Rightarrow^*_{S,A,Z} c_2$.*

Then, for any instance $I = (\nu, \mu)$ and objects $\mathbf{o} \in \nu(\mathbf{c})$ and $o \in \nu(c)$, we have $K_{I,A} \not\models \neg EQ(\tau[\mathbf{o}/\mathbf{c}], o)$.

The decidability of the sufficient condition given by Theorem 3 is not trivial since the number of superterms of τ is infinite. In the rest of this subsection, we show that only a finite number of superterms of τ have to be checked. This implies the decidability of the proposed condition.

Theorem 4. *Suppose that there is a superterm $t \in T_M(C)$ of $\tau \in T_M(C)$ satisfying neither conditions in Theorem 3. Then, there is $t' \in T_M(C)$ such that t' satisfies neither conditions in Theorem 3 and the height of t' is at most $(|P_{S,A,Z}| \cdot l_{\max})^2 + \|\tau\|$, where $l_{\max}$ is the maximum size of the terms appearing in the rules in $P_{S,A,Z}$.*

Proof. Consider a term $t \in T_M(C)$ such that $t \Rightarrow^*_{S,A,Z} c \in C$. t can be reduced to c in a various way. Choose one of the reductions from t to c, and attach a label (ρ, r') to every internal (i.e., non-leaf) occurrence r of t, where

– ρ is the rewriting rule that is applied to the subterm containing r; and
– r' is the occurrence of the left-hand side of ρ that corresponds to r.

Suppose that neither conditions in Theorem 3 are satisfied. That is, there are a term $t \in T_M(C)$ and a class $c \in C$ such that

– $\tau = t/r$ for some occurrence r;
– $t \Rightarrow^*_{S,A,Z} c_1$ for some $c_1 \in C \cup \{c_\perp\}$; and
– $t[r \leftarrow c] \Rightarrow^*_{S,A,Z} c_2$ for some $c_2 \in C \cup \{c_\perp\}$.

Label every internal occurrence of t and $t[r \leftarrow c]$ as stated above. If

– r' is a proper prefix of r'';
– r' of t and r'' of t have the same label; and
– r' of $t[r \leftarrow c]$ and r'' of $t[r \leftarrow c]$ have the same label,

then replace t/r' with t/r'' and $t[r \leftarrow c]/r'$ with $t[r \leftarrow c]/r''$. Repeat this replacement while it is possible. Let t' and $t'[r \leftarrow c]$ be the resultant terms. The height of t' is at most $(|P_{S,A,Z}| \cdot l_{\max})^2$ plus the height of τ, since the number of possible labels is at most $|P_{S,A,Z}| \cdot l_{\max}$. Moreover, t' still satisfies neither conditions in Theorem 3 since the result of the reduction of t' by $\rhd_{S,A,Z}$ is never changed by the replacement. □

6 Conclusions

In this paper, we have formalized inference of negative information under method schemas. Under the model, a polynomial-time algorithm for deciding the ISP has been proposed. Then, it has been shown that the SSP is undecidable by reducing the PCP to the SSP. A decidable sufficient condition for given negative information to be secure under any database instance of a given database schema has also been proposed.

We have already shown that for schemas with only unary methods, the proposed sufficient condition is also a necessary condition. Because of the space limitation, we have not presented this result in this paper.

The decision algorithm for the ISP is quite straightforward. There may be faster algorithms for the ISP. Also, we have shown only the decidability of the sufficient condition proposed in Section 5.2. We should investigate the time complexity for deciding the condition.

References

1. S. Abiteboul, P. Kanellakis, S. Ramaswamy, and E. Waller, "Method schemas," *Journal of Computer and System Sciences,* Vol. 51, No. 3, pp. 433–455, 1995.
2. E. Bertino and P. Samarati, "Research issues in discretionary authorizations for object bases," *Proceedings of OOPSLA-93 Conference Workshop on Security for Object-Oriented Systems,* pp. 183–199, 1994.
3. L. Chang and I.S. Moskowitz, "Bayesian methods applied to the database inference problem," *Database Security XII,* pp. 237–251, Kluwer, 1999.
4. D.E. Denning and P.J. Denning, "Data security," *ACM Computing Surveys,* Vol. 11, No. 3, pp. 227–249, 1979.
5. P.J. Downey, R. Sethi, and R.E. Tarjan, "Variations on the common subexpression problem," *Journal of the ACM,* Vol. 27, No. 4, pp. 758–771, 1980.
6. C. Farkas, T.S. Toland, and C.M. Eastman, "The inference problem and updates in relational databases," *Databases and Application Security XV,* pp. 181–194, Kluwer, 2002.
7. E.B. Fernandez, M.M. Larronodo-Peritrie, and E. Gudes, "A method-based authorization model for object-oriented databases," *Proceedings of OOPSLA-93 Conference Workshop on Security for Object-Oriented Systems,* pp. 135–150, 1993.
8. Y. Ishihara, T. Morita, and M. Ito, "The security problem against inference attacks on object-oriented databases," *Research Advances in Database and Information Systems Security,* pp. 303–316, Kluwer, 2000; A full version can be found at `http://www-infosec.ist.osaka-u.ac.jp/~ishihara/papers/dbsec99.pdf`.
9. T. Morita, Y. Ishihara, H. Seki, and M. Ito, "A formal approach to detecting security flaws in object-oriented databases," *IEICE Transactions on Information and Systems,* Vol. E82-D, No. 1, pp. 89–98, 1999.
10. M. Morgenstern, "Security and inference in multilevel database and knowledge-base systems," *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data,* pp. 357–373, 1987.
11. E. Paul, "On solving the equality problem in theories defined by Horn clauses," *Theoretical Computer Science,* Vol. 44, pp. 127–153, 1986.
12. H. Seki, Y. Ishihara and H. Dodo, "Testing type consistency of method schemas," *IEICE Transactions on Information and Systems,* Vol. E81-D, No. 3, 1998.
13. H. Seki, Y. Ishihara, and M. Ito, "Authorization analysis of queries in object-oriented databases," *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases,* LNCS 1013, pp. 521–538, 1995.
14. K. Tajima, "Static detection of security flaws in object-oriented databases," *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data,* pp. 341–352, 1996.
15. K. Zhang, "IRI: A quantitative approach to inference analysis in relational databases," *Database Security XI,* pp. 279–290, 1997.

Robust Key-Evolving Public Key Encryption Schemes[⋆,⋆⋆]

Wen-Guey Tzeng and Zhi-Jia Tzeng

Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan 30050
{tzeng,zjtzeng}@cis.nctu.edu.tw

Abstract. We propose a key-evolving paradigm to deal with the key exposure problem of public key encryption schemes. The key evolving paradigm is like the one used for forward-secure digital signature schemes. Let time be divided into time periods such that at time period j, the decryptor holds the secret key SK_j, while the public key PK is fixed during its lifetime. At time period j, a sender encrypts a message m as $\langle j, c\rangle$, which can be decrypted only with the private key SK_j. When the time makes a transit from period j to $j+1$, the decryptor updates its private key from SK_j to SK_{j+1} and deletes SK_j immediately. The key-evolving paradigm assures that compromise of the private key SK_j does not jeopardize the message encrypted at the other time periods.
We propose two key-evolving public key encryption schemes with z-resilience such that compromise of z private keys does not affect confidentiality of messages encrypted in other time periods. Assuming that the DDH problem is hard, we show one scheme semantically secure against passive adversaries and the other scheme semantically secure against the adaptive chosen ciphertext attack under the random oracle.

1 Introduction

Consider the situation that the sender Alice sends a message m to the decryptor Bob with Bob's public key PK. An attacker Carol who does not know Bob's private key SK at present time eavesdrops and records the ciphertext $c = E(PK, m)$. If Carol manages to get Bob's private key SK later on, she can get the message m no matter how much time has elapsed. The message m may carry information that is useful for a long period of time.

There are many ways to protect Bob's private key SK. One way is to replace Bob's public key when his private key is exposed. In this case every user has to update Bob's public key in its database when Bob replaces his public key. This is quite costly. Furthermore, it may not be practical since Bob may not be

⋆ Research supported in part by the National Science Council grant NSC-90-2213-E-009-152 and by the Ministry of Education grant 90-E-FA04-1-4, Taiwan, ROC.

⋆⋆ A preliminary verison appeared in the ePrint 2001/009 (February, 2001) of IACR (http://eprint.iacr.org).

aware of losing his private key. Another way is to protect Bob's private key on a secure device, such as smartcards, so that key exposure is not possible. This is quite costly, too. The third way is to use a threshold scheme to distribute SK to n trusted agents (TA's) such that k out of them can fully recover SK. When Bob receives a ciphertext c, he uses the TA's to decrypt c in a distributed way. In this case TA's not only bear heavy load of computation, but also should stay on-line always to provide decryption service. There are of course other solutions. We cannot enumerate them all.

We propose a key evolving paradigm, like the one used in forward-secure digital signature schemes [2], to deal with the key exposure problem of public-key encryption schemes. Let time be divided into periods, starting with 0. The public key PK of the decryptor Bob is fixed for the whole lifetime. Bob's private key at time period j is SK_j, $j \geq 0$. When Alice wants to send message m to Bob at time period j, she encrypts m as $\langle j, c\rangle = E(PK, j, m)$ such that one can decrypt $\langle j, c\rangle$ only with SK_j. The key evolving paradigm is that when time runs from period j to period $j+1$, Bob updates his private key from SK_j to SK_{j+1} and then deletes SK_j immediately, possibly with help from a trusted agent TA. If Carol breaks into Bob's system during time period j and gets Bob's private key SK_j at that time period, she cannot get the private keys in the other time periods directly since they have been deleted. With the key evolving paradigm, even if Bob is not aware of losing his private key, he can be sure that only those ciphertexts encrypted in the time period are exposed. In the next time period, security of newly encrypted messages are guaranteed. We consider the situation that Bob always decrypts the received messages in a short time such that the ciphertext of time period j that arrives at time period j', $j' > j$ will be discarded or re-transmitted by the sender.

Our key evolving paradigm requires that the public key be fixed for the whole lifetime. The reason is that the public key is widely distributed to other users such that it is not practical to request users check validity and timeliness of the public key frequently and on-line. Thus, a public key should remain as stable as possible. On the other hand, the decryptor is a single point. It is reasonable to allow him to interact with a trusted agent for updating his private key. Key update occurs every time period and can be finished in a prompt, which shall not incur network and computational problems to the trusted agent. The key evolving paradigm adds to the array of mechanisms of protecting secrets, such as distributed, threshold, and proactive cryptography [5,7,12,16].

A key evolving paradigm is *z-resilient* if one cannot decrypt the ciphertext $\langle j, c\rangle$ even he gets the private keys $SK_{j_1}, SK_{j_2}, \ldots, SK_{j_z}$ of z time periods, for $j \neq j_l, 1 \leq l \leq z$. If the decryptor computes his private keys with help from a trusted agent, we assume that the trusted agent is not broken in by the attacker. To ensure this security, we distribute the secret to a set of TA's and protect them with threshold and proactive cryptography.

Our results. We propose two z-resilient public-key encryption schemes such that z times of key exposures do not release any information pertinent to ciphertexts that are encrypted with non-exposed private keys. One scheme is semantically secure against passive adversaries and the other is semantically secure against the chosen ciphertext attack under the random oracle model. The size of a public key is independent of the total number of time periods, but dependent on resilience. On the other hand, the size of a private key is a constant. Without counting the pre-computation time for each time period, both encryption and decryption operations take 2 modular exponentiations. The pre-computation time is independent of time period, but dependent on resilience. Therefore, our schemes are very efficient in key size and operations of encryption and decryption.

Though the key update of our schemes needs help from TA, nevertheless, TA can be treated as an escrow agent in case of the decryptor's emergent need to decrypt ciphertexts of previous time periods. To distribute trust of TA and provide security against transient (mobile) attackers, we propose distributed and proactive key update algorithms.

We also consider distributed decryption in our KE-ENC schemes in which the decryption key is shared among a set of decryptors. We stress that even all decryptors are broken in at time period j, only SK_j is exposed. In the next time period, the decryptors share a new SK_{j+1} and the security is restored if the adversary are transient.

Related work. Dodis etc. [8] propose a similar key-evolving notion, called key-insulation, in which the decryptor updates its secret key SK_i with a physically protected central device for each key update. They consider the adaptive adversary that can corrupt the system to obtain secret keys SK_i at any time during attack. Lu and Shieh [13] consider the key-evolving protocols in the secret-key setting. The adversary they consider is weaker and the security requirement is only that the adversary cannot fully determine an un-exposed key SK_i.

2 Definitions and Preliminaries

We provide definitions for a key-evolving public-key encryption scheme and discuss its security under various security models. In the definition, we may assume that there is a trusted agent TA, who holds some secret for updating private keys of the decryptor.

Definition 1. *A* key-evolving public key encryption *scheme* KE-ENC *consists of four algorithms* $\langle KG, UPD, E, D \rangle$*:*

1. *Key generation algorithm* KG*: it is a probabilistic polynomial-time algorithm that takes as input a security parameter n and possibly other parameters and returns a* base public key PK *and corresponding base private key* SK_0*. That is,* $KG(1^n) = (PK, SK_0, s)$*, where s is the system secret (trapdoor). If TA is involved, KG distributes s to it; otherwise, KG simply discards s.*
2. *Private key update algorithm* UPD*: it takes an input the public key PK, the private key* SK_{j-1} *of time period* $j-1, j \geq 1$*, and outputs the new private key* SK_j *of time period j. That is,* $UPD(PK, SK_{j-1}, j) = SK_j$*. If TA is involved, UPD interacts with TA to compute* SK_j*.*

3. *Encryption algorithm* E*: it takes as input the base public key PK, a message m, and the time-period indicator j and outputs the ciphertext c of m at time period j. We use $E(PK, m, j) = \langle j, c\rangle$ to denote the encryption. E might be probabilistic.*
4. *Decryption algorithm* D*: it takes as input the ciphertext $\langle j, c\rangle$ of time period j and the private key $SK_{j'}$ of time period j' and outputs m if and only if $j = j'$. That is, $D(SK_j, E(PK, m, j)) = m$.*

We set no limit on the number of time periods. Key evolving can continue till the limit set by the security parameter n. The maximum number of time periods is 2^n.

We assume a single TA for simplicity. In practicality, we distribute trust to multiple trusted agents such that each TA_i holds a share s_i of the system secret s. The decryptor with private key SK_{j-1} and the TA's together can compute SK_j in a secure way, such as, through the secure multi-party computation. We discuss this in Section 4.

Operation. The system first generates a key pair (PK, SK_0) to a decryptor by the key generation algorithm KG. If TA is used, the system also sets TA for interacting with the decryptor to update the private key at the end of each time period. The public key PK is treated identically to that in a standard encryption model for registration, distribution, revocation, etc. The time is divided into periods, starting at 0, which is some reference point of time, say, January 1, 2000 and each time period covers a week. Time periods serve as "time-stamps", which are known to all, that is, every body knows the number for the current time period. At the end of each time period $j-1$, the decryptor runs the algorithm UPD, possible with TA, to update his private key SK_{j-1} of time period $j-1$ to SK_j of time period j. The decryptor need immediately delete SK_{j-1} after getting SK_j for itself's protection. Therefore, an attacker who breaks in at time period j can get the decryptor's private key SK_j only. When a user wants to send a message m to the decryptor at time period j, he computes $E(PK, m, j) = \langle j, c\rangle$ and sends it to the decryptor. Note that the time period is part of the ciphertext. When the decryptor gets $\langle j, c\rangle$, he uses his private key SK_j to decrypt the ciphertext. It may be that the decryptor gets $\langle j, c\rangle$ at time period j', $j' > j$. In this case, the decryptor cannot decrypt it. Therefore, KE-ENC is better used for applications that decryption is done in a short period of time.

The key evolving scheme cannot guarantee the desired security if the decryptor does not delete the old private key after getting the new one. It would be better to ensure erasure of old private keys by some system mechanism.

Security. We consider the conventional security models for public-key encryption schemes. We first consider the attack from passive adversaries and then the adaptive chosen ciphertext attack, which is the strongest attack against an encryption scheme.

Let KG, E and D be the key generation, encryption and decryption algorithms of a public key encryption scheme. Also, let PK and SK be the public and private keys, respectively. A public key encryption scheme is *semantically secure against passive adversaries* if a passive adversary $\mathcal{A}$, which is probabilistic polynomial-time, cannot distinguish the ciphertexts of any two messages m_0 and m_1 with a non-negligible advantage. In particular, for any passive adversary $\mathcal{A}$,

$$\Pr[KG(1^n) \rightarrow (PK, SK) : \mathcal{A}(PK, E(PK, m_b)) = b] = 1/2 + \epsilon(n),$$

where $\epsilon(n)$ is negligible and the probability is taken over b and the random coin tosses of KG, E and $\mathcal{A}$.

The adaptive chosen ciphertext attack on an encryption scheme works as follows [15]. An adversary $\mathcal{A}$ of the attack has two probabilistic polynomial-time algorithms A_1 and A_2. A_1 takes as input PK, makes some queries to the decryption oracle adaptively, and outputs two messages m_0 and m_1. Then, the encryption oracle randomly chooses a bit b and encrypts m_b as $c = E(PK, m_b)$. A_2 takes as input PK, c, m_0 and m_1, makes some queries to the decryption oracle DO in an adaptive way, and outputs b'. The decryption oracle DO takes as an input a ciphertext c', other than c, and returns its corresponding plaintext m'. If c' is invalid, DO outputs '?'. A public key encryption scheme is *semantically secure against the adaptive chosen ciphertext attack* if, for any adversary $\mathcal{A} = (A_1, A_2)$,

$$\begin{gathered}\Pr[KG(1^n) \rightarrow (PK, SK); A_1^{DO}(PK) \rightarrow (m_0, m_1) : \\ A_2^{DO}(PK, E(PK, m_b)) = b] = 1/2 + \epsilon(n),\end{gathered}$$

where $\epsilon(n)$ is negligible and the probability is taken over b and all coin tosses of KG, E, A_1 and A_2.

For the resilience property, we consider the security of of the scheme after the attacker gets some private keys additionally.

Definition 2 (Resilience). *Assume a security model for the public-key encryption scheme. A key evolving public-key encryption scheme* KE-ENC*=(KG, UPD, E, D) is* z-resilient *if the attacker cannot break the encryption scheme under the assumed security model even if he gets z private keys SK_{j_1}, SK_{j_2},..., SK_{j_z}.*

Since the attacker gets z private keys, breaking does not include obtaining the corresponding plaintext of a ciphertext that is encrypted with any of the private keys.

The *random oracle model* assumes that hash functions used in a scheme are "truly random" hash functions [3]. Although security under the random oracle model is not rigid, it does provide satisfactory security argument to related schemes in most cases [4].

Assumption. We need a standard assumption of solving the decisional discrete logarithm (DDH) problem. Let G_q be a group of a large prime order q. Consider the following two distribution ensembles R and D:

- $R = (g_1, g_2, u_1, u_2) \in G_q^4$, where g_1 and g_2 are generators of G_q;
- $D = (g_1, g_2, u_1, u_2)$, where g_1 and g_2 are generators of G_q and $u_1 = g_1^r$ and $u_2 = g_2^r$ for $r \in Z_q$.

The *DDH problem* is to distinguish the distribution ensembles R and D. That is, we would like to find a probabilistic polynomial-time algorithm $\mathcal{A}$ such that

$$|\Pr[\mathcal{A}(R_n) = 1] - \Pr[\mathcal{A}(D_n) = 1]| = \epsilon(n)$$

is non-negligible, where R_n and D_n are the size-n distributions of R and D, respectively.

A *trapdoor one-way permutation* is a probabilistic polynomial-time algorithm $\mathcal{G}$ that takes as input 1^n and outputs a triple of algorithms (f, f^{-1}, d), where f and f^{-1} are inverses of each other and deterministic polynomial-time algorithms and d is a probabilistic polynomial-time algorithm. The range of $d(1^n)$ is a subset of $\{0,1\}^k$ and f and f^{-1} on the range of $d(1^n)$ are permutations. Furthermore, for any probabilistic polynomial-time algorithm $\mathcal{A}$,

$$\epsilon(n) = \Pr[\mathcal{G}(1^n) \to (f, f^{-1}, d); d(1^n) \to x; f(x) \to y : \mathcal{A}(f, d, y) = x]$$

is negligible. For convenience, we shall call f, not $\mathcal{G}$, a "trapdoor one-way" permutation.

3 Our Protocols

We first propose a Ke-Enc scheme, based on the discrete logarithm problem, with z-resilience and show that the scheme is semantically secure against passive adversaries. We then modify the scheme to become semantically secure against the adaptive chosen ciphertext attack under the random oracle model.

3.1 Ke-Enc against Passive Adversaries

Our Ke-Enc scheme with z-resilience and security against passive adversaries is Scheme 1, which is shown in Figure 1. The main idea is to treat $f(j)$ as the time stamp. The public key $g^{f(j)} = \prod_{i=0}^{z} (g^{a_i})^{j^i}$ at time period j can be computed from the base public key $PK = \langle g^{a_0}, g^{a_1}, \ldots, g^{a_z} \rangle$. Key updating involves in computing $f(j)$ from $f(j-1)$ with help from TA. We use the ElGamal-type encryption scheme for encryption and decryption.

Correctness. The correctness of decryption follows easily since

$$s/\alpha^{f(j)} = m \cdot \prod_{i=0}^{z} (g^{a_i})^{kj^i} / g^{kf(j)} = m \cdot g^{kf(j)} / g^{kf(j)} = m.$$

Efficiency. In each time period j, we can pre-compute

$$\prod_{i=0}^{z} (g^{a_i})^{j^i} = g^{a_0}(g^{a_1}(g^{a_2}(\cdots)^j)^j)^j = g^{f(j)}.$$

It needs $z+1$ modular exponentiations and z modular multiplications, which is independent of the time period j. After pre-computation, each encryption takes

1. Algorithm KG$(1^n, z)$:
   (a) Randomly select an n-bit prime $p = 2q + 1$, where q is also a prime. All operations work over Z_p except being stated otherwise. Let G_q be the subgroup of order q in Z_p^* and g be a generator of G_q.
   (b) Randomly select a degree-z polynomial $f(x) = \sum_{i=0}^{z} a_i x^i \bmod q$.
   (c) Set the public key and base private key as
   $$PK = \langle g^{a_0}, g^{a_1}, \dots, g^{a_z} \rangle \text{ and } SK_0 = \langle f(0) \rangle.$$
   (d) Let TA hold $f(x_j)$, for some random $x_j \in Z_q, 1 \le j \le z$.
2. Algorithm UPD(PK, SK_{j-1}): the decryptor Bob and TA together compute $SK_j = f(j)$ from their shares in a secure distributed way.
3. Algorithm E(PK, m, j): randomly select $k \in Z_q$, compute
   $$\alpha = g^k, \quad s = m \cdot \prod_{i=0}^{z} (g^{a_i})^{kj^i} = m \cdot g^{kf(j)}$$
   and return the ciphertext $\langle j, \alpha, s \rangle$.
4. Algorithm D$(SK_j, \langle j, \alpha, s \rangle)$: compute and return $m = s/\alpha^{f(j)}$.

Fig. 1. Scheme 1 – discrete logarithm based KE-ENC with z-resilience and semantic security against passive adversaries.

2 modular exponentiations and 1 modular multiplication only. Each decryption takes 1 modular exponentiation and 1 modular division. Therefore, our scheme is very efficient in computation.

The public key consists of $(z + 1)$ n-bit values and the private key consists of one n-bit value only, which are both independent of the total number of time periods. The number of time periods can be as large as 2^n.

Security. We now show that Scheme 1 is semantically secure against passive adversaries even the decryptor's system is broken in z times.

Theorem 1. *Assume that the DDH problem is hard. For Scheme 1, given public key* $PK{=}\langle g^{a_0}, g^{a_1}, \dots, g^{a_z} \rangle$ *and* z *private keys* $SK_{j_1}, SK_{j_2}, \dots, SK_{j_z}$*, no probabilistic polynomial-time adversary can distinguish the ciphertexts of any two messages* m_0 *and* $m_1 \in G_q$ *at time period* j*,* $j \ne j_l, 1 \le l \le z$.

3.2 KE-ENC against Adaptive Chosen Ciphertext Attack

We modify Scheme 1 to become semantically secure against the adaptive chosen ciphertext attack under the random oracle model. The idea is to use randomness of hash functions. We construct a (probabilistic) trapdoor one-way permutation

$$h_{j,y}(k, r) = (g^k, r \cdot y^k, k \oplus H(j, r))$$

1. Algorithm KG$(1^n, z)$:
   (a) Randomly select an n-bit prime $p = 2q + 1$, where q is also a prime. Let G_q be the subgroup of order q in Z_p^* and g be a generator of G_q.
   (b) Randomly select a degree-z polynomial $f(x) = \sum_{i=0}^{z} a_i x^i \bmod q$.
   (c) Set the public key and base private key as
   $$PK = \langle g^{a_0}, g^{a_1}, \ldots, g^{a_z} \rangle \text{ and } SK_0 = \langle f(0) \rangle.$$
   (d) Let TA hold $f(x_j)$, $x_j \in Z_q$, $1 \le j \le z$.
2. Algorithm UPD(PK, SK_{j-1}): the decryptor Bob and TA together compute $SK_j = \langle f(j) \rangle$ from their shares in a secure distributed way.
3. Algorithm E$^{H_1,H_2,H_3}(PK, m, j)$: randomly select $k \in Z_q$ and $r \in G_q$, compute
   $$\alpha = g^k,\ \beta_1 = r \cdot (\prod_{i=0}^{z} (g^{a_i})^{j^i})^k = r \cdot g^{f(j)\cdot k},$$
   $$\beta_2 = k \oplus H_1(j, r),\ s = m \oplus H_2(j, r, k),\ h = H_3(j, r, k, m),$$
   and return the ciphertext $\langle j, \alpha, \beta_1, \beta_2, s, h \rangle$.
4. Algorithm D$^{H_1,H_2,H_3}(SK_j, \langle j, \alpha, \beta_1, \beta_2, s, h \rangle)$:
   (a) Compute $r = \beta_1 / \alpha^{f(j)}$, $k = \beta_2 \oplus H_1(j, r)$ and $m = s \oplus H_2(j, r, k)$.
   (b) Check whether $\alpha = g^k$ and $h = H_3(j, r, k, m)$. If it is so, return m; otherwise, return '?'.

Fig. 2. Scheme 2 – discrete logarithm based KE-ENC with z-resilience and semantic security against the adaptive chosen ciphertext attack under the random oracle model.

for time period j, where $y = g^{f(j)}$ and the trapdoor is $f(j)$. The message is encrypted with one-time pad $H_2(j, r, k)$. The hash value $H_3(j, r, k, m)$ forces the querist to be aware of m.

The modified scheme, Scheme 2, is shown in Figure 2, in which H_1, H_2, H_3 are collision-resistant hash functions with output length dependent on n.

Correctness. We can see that

$$\beta_1/\alpha^{f(j)} = r \cdot g^{kf(j)}/g^{kf(j)} = r,$$
$$\beta_2 \oplus H_1(j, r) = (k \oplus H_1(j, r)) \oplus H_1(j, r) = k, \text{ and}$$
$$s \oplus H_2(j, r, k) = (m \oplus H_2(j, r, k)) \oplus H_2(j, r, k) = m.$$

Efficiency. Encryption and decryption computation is similar to that of Scheme 1. After pre-computation, each encryption takes 2 modular exponentiations, 1 modular multiplication and 3 hash operations. Each decryption takes 1 modular exponentiation, 1 modular division and 3 hash operations. Again, computation time is independent of the time period j. Therefore, this scheme is as efficient as Scheme 1.

Again, Scheme 2's public key consists of $(z+1)$ n-bit values and private key consists of only one n-bit value, which are both independent of the total number of time periods.

Security. Assume the random oracle model, which postulates that H_1, H_2 and H_3 are "truly random" hash functions. We show that Scheme 2 is semantically secure against the adaptive chosen ciphertext attack.

Theorem 2. *Assume the random oracle model and that the discrete logarithm problem is hard. For Scheme 2, given public key $PK = \langle g^{a_0}, g^{a_1}, \ldots, g^{a_z}\rangle$ and z private keys $SK_{j_1}, SK_{j_2}, \ldots, SK_{j_z}$, no probabilistic polynomial-time adversary with access to the decryption oracle is able to find m_0 and m_1 in G_q such that their ciphertexts at time period j, $j \neq j_l, 1 \le l \le z$, are computationally distinguishable.*

4 Key Evolving with TA

In our schemes, key evolving is done with help from the TA, which holds $f(x_1), f(x_2), \ldots, f(x_z)$. We consider distributed and proactive methods to evolve private keys of the decryptor.

4.1 Distributing TA's Secret

Assume that there are z TA's and each TA_i holds a share $f(x_i)$, $1 \le i \le z$, where x_i's are distinct and large enough so that the maximum time period never reaches them. At time period $j-1$, the decryptor Bob holds $SK_{j-1} = f(j-1)$. Bob and TA's would like to compute $SK_j = f(j)$, which shall be known to Bob only. Assume that each pair of Bob and TA's share a private channel by which secret information can be passed between them.

We treat Bob as TA_0 and let $x_0 = j-1$. By the Lagrange interpolation method, the polynomial that passes shares of Bob and TA's is

$$f(x) = \sum_{k=0}^{z} (f(x_k) \cdot \prod_{0 \le i \neq k \le z} \frac{x - x_i}{x_k - x_i}).$$

TA_k can compute $s_k = f(x_k) \cdot \prod_{0 \le i \neq k \le z} \frac{j - x_i}{x_k - x_i}$, $0 \le k \le z$. Therefore, $f(j) = \sum_{k=0}^{z} s_k$. Our goal is that TA's together compute $f(j)$ and only TA_0 knows $f(j)$. Furthermore, each TA_i does not reveal any information about its share $f(x_i)$. Note that $x_0, x_1, \ldots, x_z$ are known to all TA's. The distributed protocol D-Upd for computing $f(j)$ securely is as follows.

1. Each TA_l, $1 \le l \le z$, selects a degree-z polynomial $h_l(x) = \sum_{i=1}^{z} a_{l,i} x^i + s_l$ over Z_q and sends $h_l(x_i)$ to TA_i, $0 \le i \le z$, via the private channel between them. Let $F(x) = \sum_{i=1}^{z} h_i(x)$.
2. Each TA_l, $0 \le l \le z$, computes its share $F(x_l) = \sum_{i=1}^{z} h_i(x_l)$.
3. Each TA_l, $1 \le l \le z$, sends $F(x_l)$ to TA_0 via the private channel between them.

4. TA_0 then computes the constant coefficient $\sum_{i=1}^{z} s_i$ of $F(x)$ from $F(x_0)$, $F(x_1)$,..., $F(x_z)$ by the Lagrange interpolation method and his private key $SK_j = f(j) = s_0 + \sum_{i=1}^{z} s_i$ at time period j.

Correctness follows the protocol easily. In the protocol, each TA_i does not have any information about another TA_j's share $f(x_j)$.

We can make the computation verifiable by letting each TA_l publish $g^{a_{l,0}}$, $g^{a_{l,1}}$,..., $g^{a_{l,z}}$ [10]. Each TA_i, $0 \le i \le z$, verifies whether he receives the right share $h_l(x_i)$ from TA_l, $1 \le l \le z$, by checking $g^{h_l(x_i)} = \prod_{k=0}^{z} g^{a_{l,k} {x_i}^k}$.

4.2 Proactivizing TA's Shares

We use proactive cryptography to protect TA's shares further. Let PS_i, $1 \le i \le n$, be n proactive servers. In practicality, we can make TA's as proactive servers. We proactivize each TA_i's share $f(x_i)$ into PS_i's. by the proactive (t, n)-secret sharing scheme [11]. Each evolving time period $j-1$ is divided into sub-periods $p_1, p_2, \ldots, p_b$, where $p_1, p_2, \ldots, p_{b-1}$ are refresh sub-periods in which the shares of $f(x_1), f(x_2), \ldots, f(x_z)$ are refreshed among PS_i's. In the last sub-period p_b, $f(j-1)$ of the decryptor is updated to $f(j)$.

For simplicity, we let $t = n$. Let $r_i(x)$ be a degree-(n-1) polynomial over Z_q with constant coefficient $f(x_i)$, $1 \le i \le z$. Each PS_j, $1 \le j \le n$, holds a share $r_i(j)$ for the share $f(x_i)$, $1 \le i \le z$, and refreshes them in every sub-time period p_i, $1 \le i \le b-1$, as follows.

1. Each PS_l, $1 \le l \le n$, selects n degree-(n-1) polynomials $r_{l,i}(x)$ over Z_q, $1 \le i \le n$, whose constant coefficients are all 0. PS_l sends $r_{l,i}(j)$, $1 \le i \le n$, to PS_j, $1 \le j \le n$, via the private channel between them.
2. After receiving shares from other proactive servers, PS_l, $1 \le l \le n$, updates its shares to $r'_i(l) = r_i(l) + \sum_{j=1}^{n} r_{j,i}(l)$.

To update the decryptor Bob's decryption key $f(j-1)$, we assume Bob as PS_0 and $x_0 = j-1$. Let

$$\rho_{j,k} = \prod_{0 \le i \ne k \le z} \frac{j - x_i}{x_k - x_i} \text{ and } \lambda_l = \prod_{1 \le i \ne l \le n} \frac{-i}{l-i},$$

where $0 \le k \le z$ and $1 \le l \le n$. We have

$$\begin{aligned} f(j) = \sum_{k=0}^{z} \rho_{j,k} f(x_k) &= \sum_{k=1}^{z} \sum_{l=1}^{n} \rho_{j,k} \lambda_l r_k(l) + \rho_{j,0} f(x_0) \\ &= \sum_{l=1}^{n} (\sum_{k=1}^{z} \rho_{j,k} \lambda_l r_k(l)) + \rho_{j,0} f(x_0). \end{aligned}$$

Let $s_l = \sum_{k=1}^{z} \rho_{j,k} \lambda_l r_k(l)$, which can be computed by PS_l, $1 \le l \le n$. Our proactive key update scheme P-UPD for computing $SK_j = f(j)$ in the sub-period p_b of time period j is as follows.

1. Each PS_l, $1 \leq l \leq n$, selects a degree-n polynomial $h_l(x) = \sum_{i=1}^{n} a_{l,i}x^i + s_l$ over Z_q and sends $h_l(x_i)$ to PS_i, $0 \leq i \leq n$, via the private channel between them. Let $F(x) = \sum_{i=1}^{n} h_i(x)$.
2. Each PS_l, $0 \leq l \leq n$, computes its share $F(x_l) = \sum_{i=1}^{n} h_i(x_l)$.
3. Each PS_l, $1 \leq l \leq n$, sends $F(x_l)$ to PS_0 via the private channel between them.
4. PS_0 then computes the constant coefficient $\sum_{l=1}^{n} s_l$ of $F(x)$ from $F(x_0)$, $F(x_1)$,..., $F(x_n)$ by the Lagrange interpolation method and his private key $SK_j = f(j) = \rho_{j,0}f(x_0) + \sum_{l=1}^{n} s_l$.

Correctness follows the above equations. Again, PS_i does not have any information about another PS_j's shares $r_1(j), r_2(j), \ldots, r_z(j)$. We can also make the computation verifiable by the verifiable secret sharing method as that in Section 4.1.

5 Distributed Ke-Enc Schemes

It is sometimes desirable to have distributed decryption in which the decryption key is shared among a set of decryptors B_i, $1 \leq i \leq n$.

We assume that each decryptor B_i holds a share s_i of the secret key $SK_j = f(j)$ via a polynomial $t(x) = \sum_{k=1}^{z} a_i x^i + f(j) \bmod q$ such that $s_i = t(i)$. For Scheme 1, on receiving a ciphertext (α, s), B_i computes the partial plaintext $m_i = \alpha^{s_i} \bmod p$. With $z+1$ partial plaintexts $m_{i_1}, m_{i_2}, \ldots, m_{i_{z+1}}$, one can computes the plaintext

$$m = s/\prod_{k=1}^{z+1}(m_{i_k})^{\lambda_k} \bmod p = s/\alpha^{f(j)} \bmod p,$$

where λ_k's are appropriate Lagrange coefficients. This method can be used in Scheme 2 also since knowing $\alpha^{f(j)} \bmod p$ is sufficient to decrypt the ciphertext encrypted at time period j.

6 Conclusion

We have proposed two Ke-Enc schemes to deal with the key exposure problem of public key cryptosystems. Our schemes are semantically secure against passive adversaries under the standard model and the adaptive chosen ciphertext attack under the random oracle model, respectively. To distribute trust and provide security against transient attackers, we propose distributed key update D-Upd and proactive key update P-Upd schemes.

The main drawback of these schemes is that key update needs help from TA if we want to keep the decryptor's storage a conatant. It would be interesting to find a Ke-Enc scheme, in which key update can be done by the decryptor alone in a single-decryptor mode.

References

1. M. Abdalla, L. Reyzin, "A new forward-secure digital signature scheme", *Proceedings of Advances in Cryptology – Asiacrypt 2000*, Lecture Notes in Computer Science 1976, pp.116-129, Springer-Verlag, 2000.
2. M. Bellare, S.K. Miner, "A forward-secure digital signature scheme", *Proceedings of Advances in Cryptology – Crypto 99*, Lecture Notes in Computer Science 1666, pp.431-448, Springer-Verlag, 1999.
3. M. Bellare, P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols", *Proceedings of the First ACM Conference on Computer and Communications Security*, pp.62-73, 1993.
4. R. Canetti, O. Goldreich, S. Halevi, "The random oracle methodology revisited", *Proceedings of the 30th ACM Annual Symposium on Theory of Computing*, pp.209-218, 1998.
5. R. Canetti, R. Gennaro, A. Herzberg, D. Naor, "Proactive security: long-term protection against break-ins", *CryptoBytes* 3(1), 1997.
6. R. Cramer, V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack", *Proceedings of Advances in Cryptology – Crypto '98*, Lecture Notes in Computer Science 1462, pp.13-25, Springer-Verlag, 1998.
7. Y. Desmedt, Y. Frankel, "Threshold cryptosystems", *Proceedings of Advances in Cryptology – Crypto 89*, Lecture Notes in Computer Science 435, pp.307-315, Springer-Verlag, 1989.
8. Y. Dodis, J. Katz, S. Xu, M. Yung, "Key-insulated public key cryptosystem", *Proceedings of Advances in Cryptology – Eurocrypt 02*, Lecture Notes in Computer Science 2332, pp.65-82, Springer-Verlag, 2002.
9. T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory* 31(4), pp.469-472, 1985.
10. P. Feldman, "A practical scheme for non-interactive Verifiable secret sharing", *Proceedings of the 28th IEEE Annual Symposium on the Foundations of Computer Science*, pp.427-437, 1987.
11. A. Herzberg, S. Jarcki, H. Krawczyk, M. Yung, "Proactive secret sharing, or how to cope with perpetual leakage", *Proceedings of Advances in Cryptology – Crypto 95*, Lecture Notes in Computer Science 963, pp.339-352, Springer-Verlag, 1995.
12. I. Ingemarsson, G.J. Simmons, "A protocol to set up shared secret schemes without the assistance of a mutualy trusted party", *Proceedings of Advances in Cryptology – Eurocrypt 90*, Lecture Notes in Computer Science 473, pp.266-282, Springer-Verlag, 1990.
13. C.-F. Lu, S. Shieh, "Secure key-evolving protocols for discrete logarithm schemes", Proceedings of CT-RSA 02, pp.300-310, 2002.
14. T. Pedersen, "A threshold cryptosystem without a trusted party", *Proceedings of Advances in Cryptology – Eurocrypt 91*, Lecture Notes in Computer Science 547, pp.522-526, Springer-Verlag, 1991.
15. C. Rackoff, D. Simon, "Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack", *Proceedings of Advances in Cryptology – Crypto 91*, Lecture Notes in Computer Science 576, pp.433-444, Springer-Verlag, 1991.
16. A. Shamir, "How to share a secret", *Communications of the ACM* 22(11), pp.612-613, 1979.

A Group Signature Scheme Committing the Group

Toru Nakanishi, Masayuki Tao, and Yuji Sugiyama

Department of Communication Network Engineering
Faculty of Engineering, Okayama University
3-1-1 Tsusimanaka, Okayama 700-8530, Japan
nakanisi@cne.okayama-u.ac.jp

Abstract. In this paper, a group signature scheme committing the group itself is proposed. In normal group signature schemes, a group member can anonymously sign a message on behalf of a group, and the anonymity of the signature can be revoked by only a designated authority. Consider a situation that multiple groups exist. In the proposed scheme, the signature hides not only the identity of the signer, but also the identity of the group. The group can be specified by only the designated authority. This characteristic is useful for a user's committing his attribute in the application to anonymous statistical surveys of users' attributes. Another application is the anonymous authentication of the membership with the expiration.

Keywords: Group signature scheme, Strong RSA assumption, Signature of knowledge

1 Introduction

A *group signature scheme* allows a group member to anonymously sign a message on behalf of a group, where, in addition, a membership manager and a revocation manager participate. The membership manager has the authority to add a user into the group, and the revocation manager has the authority to revoke the anonymity of a signature. Because the scheme allows us to anonymously verify user's ownership of some privilege, it is applied to various security protocols such as anonymous e-cash[1], bidding[2], and statistical survey of users' attributes [3]. In the other hand, various group signature schemes are also proposed[4,5,6,7], with the improvement of efficiency and convenience. The key scheme is proposed in [5], where the efficiency of the public key and signatures is independent from the group size. The followers [6,7] also have this good characteristic.

This paper proposes a group signature scheme with a new characteristic, which is the ability to commit the group itself. Consider the situation that the membership manager controls multiple groups. In the proposed scheme, a signature does not reveal the specified group to which the signer belong, though it reveals the fact that he belongs to some group. Only the revocation manager can specify the group from the signature. The proposed scheme is extended from one in [6], which is provably secure against the adaptive adversary, under the strong RSA assumption. Furthermore, the overhead of the addition to original scheme is small.

An application of the proposed scheme is the anonymous survey to generate the statistics of users' attributes such as gender and age. An anonymous statistical survey system of attributes is proposed in [3], where a service provider can obtain the statistics on the users' attributes. Under the assumption that some TTPs (trusted third parties) exist, this system produces the correct statistics and reveals no useful information beyond the statistics. However, the complexity of the TTP's generating the statistics depends on the number of all the users registering with the system. By using the proposed group signature scheme, we can construct the survey system that is independent from the number of registering users: A user belonging to a group correspondent with his attribute (e.g., male) sends the provider his group signature. When the distributor requests the TTPs to generate the statistics for the received signatures, the TTPs cooperatively compute the statistics by shuffling the signatures and revealing the groups from them. The detail is shown afterwards.

Another application is the anonymous authentication of the membership with the expiration. In the normal group signature, it is hard to set the different expiration for each group member. If the members with the same expiration of the membership forms a group, the normal scheme allows a verifier to check the expiration. However, this solution may weaken the anonymity, since each group becomes small. By using the group signature scheme committing the group, this problem can be solved, since the group ID correspondent with the expiration is concealed. This detail is also shown afterwards.

2 Model

We show a model of group signature scheme committing the group. Though this model describes a single revocation manager, it is easily extended into one with the multiple managers.

Definition 1. *A group signature scheme committing the group consists of the following procedures:*

Setup: *The membership manager and revocation manager generate the general public key, which is linked to multiple groups, and their secret keys.*

Join: *The membership manager issues a membership certificate for a membership secret chosen by a user joining some group. Note that the certificate assures that the user belongs to the specified group.*

Sign: *Given a message, a group member with a membership secret and its membership certificate generates the signature for the message w.r.t. the general public key.*

Verify: *A verifier checks whether a signature is made by a member in one among the groups linked to the general public key.*

Open: *Given a signature, the revocation manager with his secret specifies the identity of the signer. In addition, given a signature, the revocation manager with his secret specifies the group to which the signer belongs.*

Definition 2. *A* secure *group signature scheme committing the group satisfies the following properties:*

Unforgeability: *Only a member in one among the designated groups can generate a valid signature such that, given the signature, the revocation manager can trace the signer and the group.*

Coalition-resistance: *Colluding members cannot generate a valid membership certificate that the membership manager did not generate, even if the members adaptively obtained valid certificates from the manager.*

Anonymity: *Given a signature, it is infeasible to trace the signer, nor to trace the group to which the signer belongs.*

Unlinkability: *Given two signatures, it is infeasible to determine whether the signatures ware made by the same signer, nor whether they ware made by the members in the same group.*

No framing: *Even if the membership manager, the revocation manager, and group members collude, they cannot sign on behalf of a non-involved member.*

3 Preliminaries

3.1 Assumptions

The original group signature scheme [6] and our extension are based on the strong RSA assumption[8].

Assumption 1 (Strong RSA assumption) *Let $n = pq$ be an RSA modulus, and let G be a cyclic subgroup of $\mathcal{Z}_n^*$. Then, there exists a probabilistic polynomial-time algorithm $\mathcal{K}$ that on input $|n|$ outputs n and $z \in G$ such that, for all probabilistic polynomial-time algorithm $\mathcal{A}$, the probability that $\mathcal{A}$ on inputs n and z outputs $e \in \mathcal{Z}_{>1}$ and $u \in G$ satisfying $z = u^e \pmod{n}$ is negligible.*

Furthermore, the original scheme and ours are based on the so-called decision Diffie-Hellman (DDH) assumption. This assumption means the infeasibility to decide whether the discrete logarithms of two random elements in G to the random bases are the same. When $n = pq$ is a secure RSA modulus (i.e., $p = 2p' + 1, q = 2q' + 1$, and p, q, p', q' are prime), let $QR(n)$ be the set of quadratic residues modulo n, that is, the cyclic subgroup of $\mathcal{Z}_n^*$ generated by an element of order $p'q'$. As well as the original, we assume that $QR(n)$ satisfies the above assumptions.

3.2 Signatures of Knowledge

As building blocks, the original scheme uses signatures converted by so-called Fiat-Shamir heuristic [9] from honest-verifier zero-knowledge proofs of knowledge, which is called as signatures of knowledge. We abbreviate them as SPKs.

The SPKs are secure in the random oracle model [10], if the underlying interactive protocols are the zero-knowledge proofs of knowledge. The SPKs are denoted as

$$SPK\{(\alpha, \beta, \ldots) : R(\alpha, \beta, \ldots)\}(m),$$

which means the signature for message m by a signer with the secret knowledge $\alpha, \beta, \ldots$ satisfying the relation $R(\alpha, \beta, \ldots)$. In this notation, the Greek letters denote the signer's secret knowledge, and other parameters denote public values.

The proofs used in the original scheme and our extension show the relations among the secret discrete logarithms in the group $QR(n)$ with unknown order. The simple SPK is one proving the knowledge of a discrete logarithm of an element in the group $QR(n)$. This is converted from a zero-knowledge proof of knowledge in [8]. Let g be the generator of $QR(n)$.

Definition 3. *An SPK proving the knowledge of a discrete logarithm of $y \in QR(n)$ to the base g on message m is denoted as*

$$SPK\{\alpha : y = g^{\alpha}\}(m).$$

In this SPK, the signer needs to convince the verifier that the elements he presents are in $QR(n)$. Hence, instead the above SPK, we use $SPK\{\alpha : y^2 = (g^2)^{\alpha}\}(m)$, since $\log_{g^2} y^2 = \log_g y$ if y is in $QR(n)$ [11]. Furthermore, this SPK can be also extended into the SPK of a representation such as $SPK\{(\alpha_1, \alpha_2, \ldots)$: $y = g_1^{\alpha_1} g_2^{\alpha_2} \cdots\}(m)$.

The next SPK is one proving the equality of two discrete logarithms. This can be obtained by simply adopting the standard SPK of the equality in the known group to the setting of the unknown group's order [6].

Definition 4. *An SPK proving the knowledge of the same discrete logarithm of $y_1, y_2 \in QR(n)$ to the base g and $h \in QR(n)$ on message m is denoted as*

$$SPK\{\alpha : y_1 = g^{\alpha} \wedge y_2 = h^{\alpha}\}(m).$$

The further SPK is one proving that the discrete logarithm lies in an interval. This type of SPK used in the original [6] is not exact. It means that the SPK can prove only the membership to a larger interval than what the discrete logarithm belongs to. On the other hand, an exact zero-knowledge proof of knowledge about the interval is proposed in [12], from which the following definition is derived. Note that the efficiency of this SPK is comparable to that of about six normal $SPK\{\alpha : y = g^{\alpha}\}$.

Definition 5. *Let $]a, b[$ be an integer interval. An SPK proving the knowledge of a discrete logarithm of $y \in QR(n)$ to the base g satisfying $\log_g y \in]a, b[$ on message m is denoted as*

$$SPK\{\alpha : y = g^{\alpha} \wedge \alpha \in]a, b[\}(m).$$

The interactive versions of these SPKs are also used. The interactive ones are denoted by substituting PK for SPK, such as $PK\{\alpha : y = g^{\alpha}\}$.

4 Proposed Scheme

4.1 Idea

The original scheme [6] is constructed informally as follows: Let a and a_0 be random public elements of $QR(n)$, where the factorization of n is known by only a membership manager. In the join, a user belonging to a group chooses his membership secret x, and obtains the membership certificate (A, e) satisfying $A = (a^x a_0)^{1/e} \pmod{n}$, where e is a prime chosen by the membership manager, and x and e lie in designated intervals Γ and Λ, respectively. Then, it is proved that, under the strong RSA assumption, no adversary except the manager can forge a certificate of an unused x, even if the adversary adaptively uses the manager to obtain the certificates. The group signature consists of a triple (T_1, T_2) and an SPK proving the knowledge of $(x \in \Gamma, A, e \in \Lambda)$ such that $A \equiv (a^x a_0)^{1/e} \pmod{n}$ and (T_1, T_2) is an ElGmal ciphertext of A w.r.t. the revocation manager's key. Note that this description includes the refinement that the commitment of e is not used, as well as [11] and [7].

In the proposed scheme, we replace the membership secret and certificate with x and (A, e) satisfying $A = (a^x a_0 a_1^E)^{1/e} \pmod{n}$, where a_1 is a random public element of $QR(n)$ and E is a public identifier of the committed group. Furthermore, E is required to lies in a designated interval Δ. Then, the signature consists of a tuple (T_1, T_2, T_3, T_4) and an SPK proving the knowledge of $(x \in \Gamma, A, e \in \Lambda, E \in \Delta)$ such that $A \equiv (a^x a_0 a_1^E)^{1/e} \pmod{n}$ and (T_1, T_2) and (T_3, T_4) are ElGmal ciphertexts of A and h^E, respectively.

Remark. In [11], Camenisch and Lysyanskaya propose an anonymous credential system with anonymity revocation. The basic part in this system is the same as the group signature scheme, and is derived from the scheme [6]. The credential system uses the similar certificate to us, which is (A, e) satisfying $A = (a^x a_0 a_1^E)^{1/e} \pmod{n}$, and, by the similar SPK, the knowledge of the certificate is proved in the zero-knowledge fashion. The main difference between our scheme and the credential system is the selection of the value E. In our scheme, E is the information agreed by both the manager and the joining user at each join protocol, since E is the group's ID. In the credential system, E is the random value chosen by a joining user, and is unknown to the manager. Because of this difference, the security proof of the paper [11] cannot be adopted. Thus, after describing the detail of the proposed scheme, we should prove the security of the certificate.

4.2 Setup

Hereafter, we denote the membership manager as MM, and denote the revocation manager as RM. Let $\in_R$ denote the uniform random selection.

The setup of the proposed scheme is similar to the original one. Let ℓ_n be a security parameter. Then, MM computes two $(\ell_n/2)$-bit primes $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are primes, and then sets $n = pq$. MM also chooses $a, a_0, a_1, g, h \in_R QR(n)$. He publishes (n, a, a_0, a_1, g, h) as the public key, and keeps (p, q) as the secret key. RM chooses a secret key $x_{RM} \in_R \{0,1\}^{2\ell_n}$ and publishes the public key $y = g^{x_{RM}} \bmod n$. The interval $\Gamma =]-2^{\ell_\Gamma}, 2^{\ell_\Gamma}[$, $\Delta =]0, 2^{\ell_\Delta}[$, $\Lambda =]2^{\ell_\Lambda}, 2^{\ell_\Lambda+1}[$ are set, where $\ell_\Lambda > \ell_\Gamma + \ell_\Delta + 2$.

In addition, for the i-th group, the group's ID $E_i \in \Delta$ is selected and published, where $1 \le i \le t$ for $t \le 2^{\ell_\Delta}$ with $t = O(\text{poly}(\ell_n))$.

4.3 Join

This protocol is similar to issuing the credential in the scheme [11], which is essentially the same as the original [6]. Assume that a user U is joining a group identified by $E \in \{E_1, \ldots, E_t\}$.

1. U chooses secrets $\tilde{x} \in_R \Gamma, r_1 \in_R \{0,1\}^{2\ell_n}$, and sends MM $C_1 = g^{\tilde{x}} h^{r_1} \bmod n$. Then, U proves the correctness of C_1 to MM by
$$PK\{(\alpha, \beta) : C_1^2 = (g^2)^\alpha (h^2)^\beta\}.$$
2. MM chooses secret $\tilde{r} \in_R \Gamma$, and sends U $\tilde{r}$.
3. U computes $x = (\tilde{x} + \tilde{r} \bmod (2^{\ell_\Gamma+1} - 1)) - 2^{\ell_\Gamma} + 1$, and sends MM $C_2 = a^x \bmod n$. Note that x falls in Γ. U also computes $\dot{x} = \lfloor \frac{\tilde{x}+\tilde{r}}{2^{\ell_\Gamma+1}-1} \rfloor$, chooses $r_2 \in_R \{0,1\}^{2\ell_n}$, and sends MM $C_3 = g^{\dot{x}} h^{r_2} \bmod n$. Then, U proves the correctness of C_2 to MM by
$$\begin{aligned} PK\{(\alpha, \beta, \gamma, \delta, \epsilon, \zeta) : C_1^2 = (g^2)^\alpha (h^2)^\beta \wedge C_2^2 = (a^2)^\gamma \wedge C_3^2 = (g^2)^\delta (h^2)^\epsilon \\ \wedge C_1^2 (g^2)^{(\tilde{r} - 2^{\ell_\Gamma} + 1)} / (C_3^2)^{(2^{\ell_\Gamma+1} - 1)} = (g^2)^\gamma (h^2)^\zeta \wedge \gamma \in \Gamma\}. \end{aligned}$$
4. If the above proofs were correct, MM chooses a random prime $e \in_R \Lambda$, computes $A = (C_2 a_0 a_1^E)^{1/e} \bmod n$, and sends U the membership certificate (A, e) for the membership secret x on the group ID E.
5. U verifies that $a^x a_0 a_1^E \equiv A^e \pmod n$.

4.4 Sign and Verify

Assume that the member U has the membership secret x and the certificate (A, e) on the group ID E. To sign a message m, U chooses $w_1, w_2 \in_R \{0,1\}^{2\ell_n}$, and computes ElGmal ciphertexts $(T_1 = Ay^{w_1}, T_2 = g^{w_1})$ of A and $(T_3 = h^E y^{w_2}, T_4 = g^{w_2})$ of h^E, and the following SPK:

$$\begin{aligned} SPK\{(\alpha, \beta, \gamma, \delta, \epsilon, \zeta) : a_0^2 = (T_1^2)^\alpha (1/a^2)^\beta (1/a_1^2)^\gamma (1/y^2)^\delta \wedge T_2^2 = (g^2)^\epsilon \\ \wedge 1 = (T_2^2)^\alpha (1/g^2)^\delta \wedge T_3^2 = (h^2)^\gamma (y^2)^\zeta \wedge T_4^2 = (g^2)^\zeta \\ \wedge \alpha \in \Lambda \wedge \beta \in \Gamma \wedge \gamma \in \Delta\}(m). \end{aligned}$$

The difference from the original is the addition of (T_3, T_4) and the modified SPK. In the SPK, the first three predicates are derived from the original, and are adjusted to the relationship $A = (a^x a_0 a_1^E)^{1/e} \bmod n$. Note that the SPK is refined as the commitment of e is not used, as well as [11] and [7]. The newly added fourth and fifth predicates prove the validity of the ElGmal ciphertext

(T_3, T_4) of h^E. Though the proofs of $\alpha \in \Lambda$ and $\beta \in \Gamma$ are the same as the original, the proof of $\gamma \in \Delta$ is newly added. The last proof is required to prove the security requirement of the collision-resistance (cf. Lemma 1).

The verification of the group signature is accomplished by the verification of this SPK.

4.5 Open

To trace the identity of the member from the signature is the same as the original. That is accomplished by RM's decrypting (T_1, T_2) with the public verifiability.

To trace the group of the signature, RM decrypts (T_3, T_4) of the signature, by $T_3/(T_4)^{x_{RM}} \bmod n$. The decryption produces the value $\tilde{h} \equiv h^E \pmod{n}$, and, by checking $\tilde{h} = h^{E_j} \bmod n$ for all E_j, the corresponding group is found. RM can prove the correctness by $SPK\{\alpha : T_3^2/(h^E)^2 = (T_4^2)^\alpha \wedge y^2 = (g^2)^\alpha\}$.

5 Security

Before discussing the security of the proposed scheme, we show the following lemma on the security of the membership certificate.

Lemma 1 (Coalition-resistance). *Assume the strong RSA assumption, and that the number K of the certificates that MM issues, and the number t of the groups are polynomially bounded. Then, it is infeasible that an attacker, who is allowed to adaptively run the join protocol with MM and obtain K membership certificates, generate a membership certificate that MM did not generate.*

Proof sketch. This is derived from the proof in [6]. Let $\mathcal{M}$ be an attacker that is allowed to adaptively run the join protocol and obtain K membership certificates $(A_i, = (a^{x_i} a_0 a_1^{E_i})^{1/e_i} \bmod n, e_i)$ for $i = 1, \ldots, K$. Here, we prove that, if $\mathcal{M}$ outputs a tuple $(\hat{x}, \hat{A}, \hat{e}, \hat{E})$ such that $\hat{A} = (\hat{a}^{\hat{x}} a_0 a_1^{\hat{E}})^{1/\hat{e}}$ with $\hat{x} \in \Gamma, \hat{e} \in \Lambda, \hat{E} \in \Delta$, and $(\hat{x}, \hat{e}) \not\models (x_i, e_i)$ for all $1 \leq i \leq K$ with non-negligible probability, the strong RSA problem can be solved.

Given a pair (n, z) on the strong RSA assumption, a random one of the following two games with $\mathcal{M}$ is conducted repeatedly.

Game 1.

1. Select $x_1, \ldots, x_K \in \Gamma$, $e_1, \ldots, e_K \in \Lambda$, and $E_1, \ldots, E_t \in \Delta$.
2. Set $a = z^{\prod_{1 \leq \ell \leq K} e_\ell} \bmod n$.
3. Choose $r, r' \in_R \Gamma$ and set $a_0 = a^r$ and $a_1 = a^{r'}$.
4. For all $1 \leq i \leq K$ and all $1 \leq j \leq t$, compute
$$A_{ij} = z^{(x_i + r + r' E_j) \prod_{1 \leq \ell \leq K, \ell \neq i} e_\ell} \bmod n.$$
Note that $A_{ij} \equiv (a^{x_i} a_0 a_1^{E_j})^{1/e_i} \pmod{n}$.
5. Select $g, h \in_R QR(n), x_{RM} \in_R \{0,1\}^{2\ell_n}$, and set $y = g^{x_{RM}} \bmod n$.

6. Run the join protocol K times with $\mathcal{M}$ on input $(n, a, a_0, a_1, y, g, h, E_1, \ldots, E_t)$. Consider the i-th protocol run. Receive the commitment C_1 with the PK, and extract $\tilde{x}_i$ satisfying $C_1 \equiv g^{\tilde{x}_i} h^{r_1} \pmod{n}$ from the PK. Choose $\tilde{r}_i \in \Gamma$ satisfying $x_i = (\tilde{x}_i + \tilde{r}_i \bmod (2^{\ell_\Gamma+1} - 1)) - 2^{\ell_\Gamma} + 1$, and send $\tilde{r}_i$ $\mathcal{M}$. Continue the rest of the protocol as specified until step 4. Then, send (A_{ij}, e_i) $\mathcal{M}$ for the agreed group ID E_j.
   After the protocols are conducted K times, $\mathcal{M}$ outputs $(\hat{x}, \hat{A}, \hat{e}, \hat{E})$ with $\hat{x} \in \Gamma, \hat{e} \in \Lambda, \hat{E} \in \Delta$, and $\hat{A} \equiv (a^{\hat{x}} a_0 a_1^{\hat{E}})^{1/\hat{e}} \pmod{n}$.
7. If $\gcd(\hat{e}, e_i) \neq 1$ for some $1 \leq i \leq K$ then output $\perp$ and quit. Otherwise, let $\tilde{e} := (\hat{x} + r + r'\hat{E}) \prod_{1 \leq \ell \leq K} e_\ell$. Note that $\hat{A}^{\hat{e}} \equiv z^{\tilde{e}} \pmod{n}$. Because of $\gcd(\hat{e}, e_i) = 1$ for all $1 \leq i \leq K$, we have $\gcd(\hat{e}, \tilde{e}) = \gcd(\hat{e}, \hat{x} + r + r'\hat{E})$. Thus, the extended Euclidean algorithm can compute $\alpha, \beta \in \mathcal{Z}$ satisfying $\alpha\hat{e} + \beta\tilde{e} = \gcd(\hat{e}, \hat{x} + r + r'\hat{E})$. Therefore, by setting $u := z^\alpha \hat{A}^\beta \bmod n$ and $e := \hat{e}/\gcd(\hat{e}, \hat{x} + r + r'\hat{E})$, we have $u^e \equiv z \pmod{n}$. Because of $\hat{e} \in \Lambda$ and $\hat{x} + r + r'\hat{E} < 2^{\ell_\Gamma+1} + 2^{\ell_\Gamma} 2^{\ell_\Delta} < 2^{\ell_\Lambda}$, we have $e > 1$. Output (u, e).

Game 2.

1. Select $x_1, \ldots, x_K \in \Gamma$, $e_1, \ldots, e_K \in \Lambda$, and $E_1, \ldots, E_t \in \Delta$.
2. Choose $\tilde{\imath} \in_R \{1, \ldots, K\}$. Set $a = z^{\prod_{1 \leq \ell \leq K, \ell \neq \tilde{\imath}} e_\ell} \bmod n$.
3. Choose $r, r' \in_R \Gamma$. Compute $a_0 = a^{re_{\tilde{\imath}} - x_{\tilde{\imath}}} \bmod n$, $a_1 = a^{r' e_{\tilde{\imath}}} \bmod n$, and $A_{\tilde{\imath}j} = a^{r + r'E_j} \bmod n$ for all $1 \leq j \leq t$. Note that $A_{\tilde{\imath}j} \equiv (a^{x_{\tilde{\imath}}} a_0 a_1^{E_j})^{1/e_{\tilde{\imath}}} \pmod{n}$ for all j.
4. For all $1 \leq i \leq K$ $(i \neq \tilde{\imath})$ and all $1 \leq j \leq t$, compute
$$A_{ij} = z^{(x_i + re_{\tilde{\imath}} - x_{\tilde{\imath}} + r' e_{\tilde{\imath}} E_j) \prod_{1 \leq \ell \leq K, \ell \neq i, \tilde{\imath}} e_\ell} \bmod n.$$
   Note that $A_{ij} \equiv (a^{x_i} a_0 a_1^{E_j})^{1/e_i} \pmod{n}$.
5. Select $g, h \in_R QR(n), x_{RM} \in_R \{0,1\}^{2\ell_n}$, and set $y = g^{x_{RM}} \bmod n$.
6. Run the join protocol K times with $\mathcal{M}$ on input $(n, a, a_0, a_1, y, g, h, E_1, \ldots, E_t)$. Consider the i-th protocol run. Receive the commitment C_1 with the PK, and extract $\tilde{x}_i$ satisfying $C_1 = g^{\tilde{x}_i} h^{r_1}$ from the PK. Choose $\tilde{r}_i \in \Gamma$ satisfying $x_i = (\tilde{x}_i + \tilde{r}_i \bmod (2^{\ell_\Gamma+1} - 1)) - 2^{\ell_\Gamma} + 1$, and send $\tilde{r}_i$ $\mathcal{M}$. Continue the rest of the protocol as specified until step 4. Then, send (A_{ij}, e_i) $\mathcal{M}$ for the agreed group ID E_j.
   After the protocols are conducted K times, $\mathcal{M}$ outputs $(\hat{x}, \hat{A}, \hat{e}, \hat{E})$ with $\hat{x} \in \Gamma, \hat{e} \in \Lambda, \hat{E} \in \Delta$, and $\hat{A} \equiv (a^{\hat{x}} a_0 a_1^{\hat{E}})^{1/\hat{e}} \pmod{n}$.
7. If $\gcd(\hat{e}, e_{\tilde{\imath}}) \neq e_{\tilde{\imath}}$ then output $\perp$ and quit. Otherwise, we have $\hat{e} = se_{\tilde{\imath}}$ for some s and can define $Z := \hat{A}^s / a^{r + r'\hat{E}} \bmod n$ if $\hat{x} \geq x_{\tilde{\imath}}$ and $Z := a^{r + r'\hat{E}} / \hat{A}^s \bmod n$ otherwise. Then, from $a^{r + r'\hat{E}} \equiv (a^{x_{\tilde{\imath}}} a_0 a_1^{\hat{E}})^{1/e_{\tilde{\imath}}} \pmod{n}$, we have $Z \equiv (a^{|\hat{x} - x_{\tilde{\imath}}|})^{1/e_{\tilde{\imath}}} \equiv (z^{|\tilde{e}|})^{1/e_{\tilde{\imath}}} \pmod{n}$ with $\tilde{e} := (\hat{x} - x_{\tilde{\imath}}) \prod_{1 \leq \ell \leq K, \ell \neq \tilde{\imath}} e_\ell$. Because $\gcd(e_{\tilde{\imath}}, \prod_{1 \leq \ell \leq K, \ell \neq \tilde{\imath}} e_\ell) = 1$, we have $\gcd(e_{\tilde{\imath}}, |\tilde{e}|) = \gcd(e_{\tilde{\imath}}, |\hat{x} - x_{\tilde{\imath}}|)$. Hence, compute $\alpha, \beta \in \mathcal{Z}$ satisfying $\alpha e_{\tilde{\imath}} + \beta|\tilde{e}| = \gcd(e_{\tilde{\imath}}, |\hat{x} - x_{\tilde{\imath}}|)$. Therefore, by setting $u := z^\alpha Z^\beta \bmod n$ and $e := e_{\tilde{\imath}}/\gcd(e_{\tilde{\imath}}, |\hat{x} - x_{\tilde{\imath}}|)$, we have $u^e \equiv z \pmod{n}$. Because of $e_{\tilde{\imath}} \in \Lambda$ and $|\hat{x} - x_{\tilde{\imath}}| < 2^{\ell_\Gamma+2} < 2^{\ell_\Lambda}$, we have $e > 1$. Output (u, e).

Consequently, by repeatedly playing a random one of Game 1 or 2 until the result is not $\perp$, an attacker with the access to $\mathcal{M}$ can solve the strong RSA problem in the expected polynomial time in K and t. Since this contradicts the strong RSA assumption, no one except MM can generate a valid membership certificate.

$\square$

The next lemma shows the security of the signature generation.

Lemma 2. *The interactive version of the group signature generation is a honest-verifier zero-knowledge proof of knowledge of* (x, A, e, E) *such that* $A \equiv (a^x a_0 a_1^E)^{1/e} \pmod{n}$ *with* $x \in \Gamma, e \in \Lambda$ *and* $E \in \Delta$*. Furthermore,* (T_1, T_2) *and* (T_3, T_4) *are ElGmal ciphertexts for* A *and* $h^E \bmod n$ *w.r.t. public key* y*, respectively.*

Proof sketch. It is easy to prove the completeness and the zero-knowledgeness. The rest is to extract the knowledge. Owing to the knowledge extractor of each SPK, we can extract the values x_1, x_2, x_3, x_4, x_5 and x_6 satisfying

$$a_0 = T_1^{x_1}(1/a)^{x_2}(1/a_1)^{x_3}(1/y)^{x_4}, T_2 = g^{x_5}, 1 = T_2^{x_1}(1/g)^{x_4},$$
$$T_3 = h^{x_3}y^{x_6}, T_4 = g^{x_6},$$
$$x_1 \in \Lambda, x_2 \in \Gamma, \text{ and } x_3 \in \Delta.$$

From the second and third equations, the equation $g^{x_4} \equiv g^{x_1 x_5} \pmod{n}$ holds, and thus $(1/y)^{x_4} \equiv (1/y)^{x_1 x_5} \pmod{n}$ also holds. Therefore, $a^{x_2} a_0 a_1^{x_3} \equiv T_1^{x_1}/y^{x_4} \equiv T_1^{x_1}/y^{x_1 x_5} \equiv (T_1/y^{x_5})^{x_1} \pmod{n}$ holds. Hence, the knowledge of $(x = x_2, A = T_1/y^{x_5} \bmod n, e = x_1, E = x_3)$ satisfying $A \equiv (a^x a_0 a_1^E)^{1/e} \pmod{n}$, $e \in \Lambda, x \in \Gamma$, and $E \in \Delta$ can be extracted.

Furthermore, since, for the extracted knowledge A and $h^E \bmod n$, $T_1 \equiv A y^{x_5} \pmod{n}$, $T_2 \equiv g^{x_5} \pmod{n}$ and $T_3 \equiv h^E y^{x_6} \pmod{n}, T_4 \equiv g^{x_6} \pmod{n}$ hold, (T_1, T_2) and (T_3, T_4) are ElGmal ciphertexts for A and $h^E \bmod n$ w.r.t. public key y, respectively.

$\square$

Now, we discuss the security properties.

Unforgeability: Because of Lemma 2, it is assured that the signer knows a certificate (A, e). On the other hand, because of Lemma 1, it is infeasible for anyone except MM to compute such (A, e). Therefore, only a member can sign a message.
Furthermore, because of Lemma 2, it is assured that (T_1, T_2) and (T_3, T_4) are ElGmal ciphertexts for A and h^E of a member, respectively. Thus, the revocation manager can decrypt the ciphertexts to match the corresponding identity and group.

Coalition-resistance: This property follows from Lemma 1.

Anonymity and unlinkability: These properties follows since the group signature uses the ElGmal ciphertexts and the SPK, as well as the original.

No framing: Because of Lemma 2, signing on behalf of a member with the certificate (A, e) on the group ID E requires the knowledge of $x = \log_a A^e/a_0 a_1^E$.

On the other hand, the join protocol reveals only $C_2 = a^x$ together with (A, e), since the PK and the commitments C_1 and C_3 reveal no information. The group signature also reveals no information on x owing to the ElGmal ciphertexts and the SPK. Therefore, because of the infeasibility of computing the discrete logarithm, the other including MM cannot extract the knowledge x.

6 Applications

6.1 Application to Anonymous Statistical Survey of Attributes

In [3], the anonymous statistical survey of attributes is proposed. The purpose of this system is that a service provider collects the attributes (e.g., gender, age, and job) from the users. This is useful for working out the marketing strategy. On the other hand, the users desire to anonymously use the service, and furthermore not to offer their attributes since offering many attributes may weaken the anonymity. Therefore, in this system, the use of some TTPs allows the distributor to obtains only the statistics of the attributes. The statistics means the number of users with each attribute value. The example of the gender is the information of 50 male users and 50 female users. This system satisfies both the correctness of the statistics and the anonymity of users. The anonymity means that no useful information beyond the statistics is revealed.

The participants in this system are, a provider, users, trustees, and attribute authority. It is assumed that a quorum of the trustees are corrupted. It is assumed that the attribute authority can be convinced of users' genuine attributes, and that the authority issues the correct certificate of the attributes.

The previous system [3] uses the normal group signature, which includes an ElGamal ciphertext, and secure multiparty computations on ElGamal ciphertexts. However, the secure multiparty computations are inefficient, since the efficiency is in proportion to the number of all the users registering with the system.

When using the proposed group signature scheme committing the group, we can construct the efficient survey system easily as follows. Here, though the description with a single trustee is shown, the threshold ElGamal encryption [13] allows us to construct the version with multiple trustees.

Setup: The group signature scheme committing the group is set up, where the attribute authority is MM and the trustee is RM.

Register: To join the system, a user conducts the join protocol of the group signature scheme with the attribute authority, where the user joins the group corresponding with his attribute value (e.g., male).

Offer: During the service, the user sends the provider his group signature committing the attribute group. The provider collects the group signatures of users during a period.

Generate: The provider gives the trustee the collected signatures. The trustee shuffles the ElGamal ciphertexts (T_3, T_4) in the group signature verifiably (e.g., [14]). Then, the trustee opens the signatures to reveal the groups. The revealed groups indicate the statistics of the attribute.

Owing to the anonymity and unlikability of the group signature, and the shuffle, the provider obtains no information beyond the statistics. Owing to the unforgeability of the group signature and the verifiability of the shuffle, the correctness of the statistics is assured.

Next, we discuss the efficiency of the usually used protocols. For offering the attributes, the previous system uses the group signature [5]. The efficiency is comparable to that of the proposed group signature. In the other hand, the efficiency in the our generate protocol does not depend on the number of all the registering users, though the previous system depends on it.

6.2 Application to Anonymous Authentication of Membership with Expiration

The interactive version of the group signature (i.e., The SPK in the signature is replaced by the PK) is called the identity escrow [15], which is the interactive anonymous authentication of the membership. This scheme allows a service provider to verify whether a user requesting the service is a valid user with the access privilege, while the user's identity is concealed, where the membership of a group means the ownership of the privilege. In such an application, generally the privilege should be expired after a certain interval of time. Thus, the membership should be expired.

In the normal group signature scheme, it is difficult to set the different expiration date for each membership certificate. A plausible solution in the normal scheme is to constitute a group of the users with the same expiration date. However, this strategy weakens the anonymity, since the size of the group becomes smaller.

The proposed group signature can be extended into the scheme with the expiration as follows:

Setup: In the setup protocol, the group IDs $E_1, \ldots, E_t$ are assigned to the expiration dates $D_1, \ldots, D_t$, respectively, such that $E_i \leq E_j$ iff $D_i \leq D_j$.

Join: The join protocol is conducted for E_j assigned to the expiration date D_j of the membership certificate.

Sign and verify: At the time of signing, assume that the dates $D_1, \ldots, D_j$ are expired. Then, consider an interval Δ_j that includes $E_{j+1}, \ldots, E_t$ and that does not include $E_1, \ldots, E_j$. The signature uses Δ_j instead of Δ.
Since $E \in \Delta_j$ is proved, the verifier can be convinced that the membership certificate is not expired at that time. On the other hand, the signature does not reveal the concrete expiration date corresponding with E. Thus, the anonymity remains.

Note that the efficiency of signing and verifying is the same as that of the proposed group signature scheme.

7 Conclusion

We have proposed a group signature scheme committing the group. The collision-resistance against the adaptive adversary is proved under the strong RSA assumption. Signing in this scheme is as efficient as the original [6], since the addition is only an ElGamal ciphertext and the efficient SPK.

Furthermore we have shown two efficient applications: One is to offer the user's attribute in the anonymous statistical survey. Another is the anonymous authentication of the membership with the expiration.

References

1. T. Nakanishi and Y. Sugiyama, "Unlinkable divisible electronic cash," Proc. Third International Workshop on Information Security (ISW2000), LNCS 1975, pp.121–134, Springer–Verlag, 2000.
2. K. Sakurai and S. Miyazaki, "An anonymous electronic bidding protocol based on a new convertible group signature scheme," Proc. 5th Australasian Conference on Information Security and Privacy (ACISP2000), LNCS 1841, pp.385–399, Springer–Verlag, 2000.
3. T. Nakanishi and Y. Sugiyama, "Anonymous statistical survey of attributes," Proc. 6th Australasian Conference on Information Security and Privacy (ACISP2001), LNCS 2119, pp.460–473, Springer–Verlag, 2001.
4. D. Chaum and E. van Heijst, "Group signatures," Advances in Cryptology — EUROCRYPT '91, LNCS 547, pp.241–246, Springer–Verlag, 1991.
5. J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," Advances in Cryptology — CRYPTO '97, LNCS 1294, pp.410–424, Springer–Verlag, 1997.
6. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik, "A practical and provably secure coalition-resistant group signature scheme," Advances in Cryptology — CRYPTO 2000, LNCS 1880, pp.255–270, Springer–Verlag, 2000.
7. D.X. Song, "Practical forward secure group signature schemes," Proc. ACM Conference on Computer and Communications Security, pp.225–234, 2001.
8. E. Fujisaki and T. Okamoto, "Statistical zero knowledge protocols to prove modular polynomial relations," Advances in Cryptology — CRYPTO '97, LNCS 1294, pp.16–30, Springer–Verlag, 1997.
9. A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," Advances in Cryptology — CRYPTO '86, LNCS 263, pp.186–194, Springer–Verlag, 1987.
10. M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," Proc. First Annual Conference on Computer and Communications Security, pp.62–73, Association for Computing Machinery, 1993.
11. J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," Advances in Cryptology — EUROCRYPT2001, LNCS2045, pp.93–118, Springer–Verlag, 2001.
12. F. Boudot, "Efficient proofs that a committed number lies in an interval," Advances in Cryptology — EUROCRYPT2000, LNCS1807, pp.431–444, Springer–Verlag, 2000.
13. Y.D.Y. Frankel, "Threshold cryptosystems," Advances in Cryptology — CRYPTO'89, LNCS 435, pp.307–315, Springer–Verlag, 1990.
14. J. Furukawa and K. Sako, "An efficient scheme for proving shuffle," Advances in Cryptology — CRYPTO 2001, LNCS 2139, pp.368–387, Springer–Verlag, 2001.
15. J. Kilian and E. Petrank, "Identity escrow," Advances in Cryptology — CRYPTO '98, LNCS 1462, pp.169–185, Springer–Verlag, 1998.

Unconditionally Secure Key Insulated Cryptosystems: Models, Bounds and Constructions[⋆]

Yumiko Hanaoka[1], Goichiro Hanaoka[2], Junji Shikata[3], and Hideki Imai[2]

[1] Security Systems Laboratory, Multimedia Laboratories, NTT DoCoMo, Inc.
3-5 Hikarino-oka, Yokosuka 239-8536, Japan
claudia@mml.yrp.nttdocomo.co.jp

[2] Information & Systems, Institute of Industrial Science, University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo 153-8508, Japan
hanaoka@imailab.iis.u-tokyo.ac.jp, imai@iis.u-tokyo.ac.jp

[3] Graduate School of Environment and Information Sciences, Yokohama National University
79-7 Tokiwadai, Hodogaya-ku, Yokohama 240-8501, Japan
shikata@mlab.jks.ynu.ac.jp

Abstract. Computer systems are constantly under attack and illegal access is a constant threat which makes security even more critical. A system can be broken into and secret information, e.g. decryption key, may be exposed. Very recently, a new framework for the protection against such key exposure problem was suggested and was called, *key-insulated encryption* (KIE). In this paper, we study key insulated cryptosystems without computational assumptions. First, we define the model of KIE in unconditional setting and show lower bounds on required memory sizes. Our bounds are all tight since our concrete construction of KIE achieves the bounds. In other words, this construction is optimal in terms of memory sizes of a user, a trusted device and a sender. We then, extend the concept of it further, and add an extra property so that any pair of users in the system can communicate with each other. We called the version with this added extension, *dynamic and mutual key insulated encryption* (DMKIE), and concrete implementations of DMKIE are also shown. Finally, we discuss the relationship among KIE, *key predistribution schemes* (KPS) and *broadcast encryption schemes* (BES), specifically, showing that DMKIE can be constructed from KPS or BES.

1 Introduction

Background. The physical loss of a Personal Computer is costly, but the loss of computing power and the data stored on the computer can be disastrous. Also, lost data is very expensive to replace. Sensitive or confidential information is vulnerable to unauthorized access without the proper security, but is also an

⋆ The second author is supported by a Research Fellowship from Japan Society for the Promotion of Science (JSPS).

unavoidable part in networking life, whether it is the loss of keys, important data or files. Nonetheless, if not able to avoid such loss of secret information, we still want to suffer the least of the damage. In order to minimize the risk of loss, there has been ways to achieve this by splitting the secret information that you have into small pieces, or shares, and store each shares on different devices beforehand [16,8]. And recently, Dodis, Katz, Xu and Yung [10] suggested a new framework to the solution to the above key exposure problem, and was called, *key-insulated encryption* (KIE). KIE combines key splitting with key evolution ideas, which are often used in forward-secure schemes [3,1]. This basically means, extracting the benefit of splitting, leaving the information manageable in case of loss, and having decryption as a stand-alone user operation as well. In KIE, a user has a trusted device, e.g. smart card, and a user and a trusted device has *stage 0 user secret key* and *master helper key*, respectively. When a sender encrypts the message by using an encryption key and transmits it to another user, here, the length of lifetime of the encryption key is divided by the number of stages $i = 1, 2, \cdots, N$, where encryption in stage i is performed as a function of the encryption key, i and the message. Accordingly, decryption in stage i is performed by the user using *stage i user secret key* obtained from the following key-update process performed in the beginning of stage i: first, the trusted device sends a *stage i helper key* to the user which is computed as a function of the master helper key and i; second, the user computes the stage i user secret key as a function of the stage $i-1$ user secret key and the stage i master helper key; and finally, the user deletes the stage $i-1$ user secret key. The intended security goes like this: if the trusted device is not compromised, a user's secret keys of at least $t+1$ different stages must all be exposed in order to compromise the ciphertext encrypted for all the rest of the stages. Additionally, in order to violate the ciphertext, the user's secret key of at least one stage must be exposed in addition to exposing the trusted device.

As mentioned earlier, the first concrete implementation of KIE was proposed by Dodis, Katz, Xu and Yung [10]. Based on identity-based encryption schemes [7,9], Bellare and Palacio [2] further showed efficient constructions of KIE with optimal threshold, namely, regardless of the number of user stages that has been violated, ciphertexts of all the rest of the stages still remain secure. Up to this point, all of constructions of KIE also described here so far, are all based on computational difficulty of solving certain hard problems, e.g. discrete logarithms. There has yet, an investigation made to explore what constructions and lower bounds of required memory sizes may be for *unconditional setting* that is with no computational assumptions, which is still remained unclear.

Our Results. In this paper, we study unconditionally secure key insulated cryptosystems, specifically, on the lower bounds, optimal constructions and some extensions. We begin, by defining the model of KIE for unconditional setting and show the lower bounds of required memory sizes. We estimate that the required memory sizes for a ciphertext, a user secret key, a master helper key, a helper key and a sender's key be at least $H(M)$ bit, $(k+1)H(M)$ bit, $(k+1)(t+1)H(M)$ bit, $(k+1)H(M)$ bit and $(t+1)H(M)$ bit, respectively, where $H(M)$ is the size of a plaintext assuming that the maximum number of malicious senders is at most k, and at most t of the user's keys are exposed. These bounds are all tight since we show a concrete construction of KIE that achieves all the bounds, in order words, that, these constructions are optimal in terms of memory sizes of a user, a trusted device and a sender. It should be noticed that in our scheme, we address *strong key insulation* [10,2], which means that, even if the secure device is compromised and end up having the master helper key revealed, the adversary still cannot compute any of user's secret keys unless one of user's secret keys is exposed along with the master helper key, and is exactly the motivation of key splitting. Next, we extend the concept of KIE further and add an extra property so that any pair of users in a system can communicate having the same security benefits as existing KIE systems which are based on computational assumptions. We call the extended KIE, *dynamic and mutual key insulated encryption* (DMKIE) and show concrete implementations as well. Finally, we discuss the relationship among KIE, *key predistribution schemes* (KPS) [4,14,5,17,12] and *broadcast encryption schemes* (BES) [11,6,17]. Specifically, we show that a DMKIE can be constructed from a KPS or a BES.

2 Model and Bounds

2.1 The Model

The model of our KIE with unconditional setting includes a user U, the user's trusted device H and n senders $S_1, \cdots, S_n$. In the initial phase, U generates a stage 0 user secret key uk_0, a master helper key mk^* and senders' keys $e_1, \cdots, e_n$. These keys are distributed to their corresponding owners via secure channels. After distributing these key, U deletes $mk^*, e_1, \cdots, e_n$ from his memory. For updating user's key for stage i, H sends the user a stage i helper key, mk_i, which is computed as a function of mk^* and i. U then calculates a stage i user secret key, uk_i, as a function of uk_{i-1} and mk_i. U deletes uk_{i-1} and mk_i from his memory. Now, a sender, $S_j \in \{S_1, \cdots, S_n\}$, encrypts the message m_{ij} by e_j with encryption in stage i performed as a function of e_j, i and m_{ij}, and generates a ciphertext c_{ij}. U then recovers m_{ij} as a function of uk_i and c_{ij}. We assume that at most one message in each stage is encrypted, and that, there exist at most k malicious senders and at most t user secret keys that are exposed.

2.2 Security Definition

For the model of unconditionally secure KIE, it is necessary to satisfy the following requirements:(1) U can correctly recover m_{ij} from c_{ij} with probability 1. (2) S_j can correctly create c_{ij} from m_{ij} with probability 1. (3) uk_i can correctly be generated from uk_{i-1} and mk_i. (4) mk_i can correctly be generated from mk^*. (5) Even if mk^* is exposed, none of U's secret keys will be computed without also having one of U's secret keys. (6) Any coalition of adversaries who may have obtained t exposed user secret keys cannot obtain any information on m_{ij} from c_{ij}. (7) Any coalition of adversaries who may have mk^* cannot obtain any information on m_{ij} from c_{ij} unless they have oen of user secret keys. (8) No ciphertext provides any information regarding any plaintext. Letting $U\!K_\ell$ $(\ell = 0,1,\cdots), E_\ell$ $(\ell = 1,\cdots,n), M\!K^*, M\!K_\ell$ $(\ell = 0,1,\cdots), M_{ij}$ $(i = 0,1,\cdots,\ j = 1,\cdots,n)$ and C_{ij} $(i = 0,1,\cdots,\ j = 1,\cdots,n)$ be random variables induced by uk_ℓ $(\ell = 0,1,\cdots), e_\ell$ $(\ell = 1,\cdots,n), mk^*, mk_\ell$ $(\ell = 0,1,\cdots), m_{ij}$ $(i = 0,1,\cdots,\ j = 1,\cdots,n)$ and c_{ij} $(i = 0,1,\cdots,\ j = 1,\cdots,n)$, respectively, a (t,k,n)-secure KIE is formally defined as follows:

Definition 1 A KIE in the above model is called a (t,k,n)-*secure KIE* if the following equations hold;

$$
\begin{array}{ll}
(1) & H(M_{ij}|C_{ij}, U\!K_i) = 0 \quad (i = 0,1,\cdots,\ j = 1,\cdots,n) \\
(2) & H(C_{ij}|M_{ij}, E_j) = 0 \quad (i = 0,1,\cdots,\ j = 1,\cdots,n) \\
(3) & H(U\!K_i|U\!K_{i-1}, M\!K_i) = 0 \quad (i = 1,2,\cdots) \\
(4) & H(M\!K_i|M\!K^*) = 0 \quad (i = 1,2,\cdots) \\
(5) & H(U\!K_i|M\!K^*) = H(U\!K_i) \quad (i = 0,1,\cdots) \\
(6) & H(M_{ij}|C_{ij}, E_{\ell_1},\cdots,E_{\ell_k}, U\!K_{\kappa_1},\cdots,U\!K_{\kappa_t}) = H(M_{ij}) \\
 & (i = 0,1,\cdots,\ j = 1,\cdots,n,\ j \notin \forall\{\ell_1,\cdots,\ell_k\} \subset \{1,\cdots,n\}, \\
 & \ i \notin \forall\{\kappa_1,\cdots,\kappa_t\} \subset \{0,1,\cdots\}) \\
(7) & H(M_{ij}|M\!K^*, C_{ij}, E_{\ell_1},\cdots,E_{\ell_k}) = H(M_{ij}) \\
 & (i = 0,1,\cdots,\ j = 1,\cdots,n,\ j \notin \forall\{\ell_1,\cdots,\ell_k\} \subset \{1,\cdots,n\}) \\
(8) & H(M_{i_0j_0}, M_{i_1j_1},\cdots|C_{\delta\gamma}) = H(M_{i_0j_0}, M_{i_1j_1},\cdots) \\
 & (\forall\{i_0,i_1,\cdots\} \subseteq \{0,1,\cdots\},\ \forall\{j_0,j_1,\cdots\} \subseteq \{1,\cdots,n\}, \\
 & \ \delta = 0,1,\cdots,\ \gamma = 1,\cdots,n)
\end{array}
$$

2.3 Lower Bounds

In this subsection, lower bounds on required memory sizes for a (t,k,n)-secure KIE are estimated. First, we show a lower bound on the required memory size for a ciphertext of which can be easily obtained from Shannon's and from conditions (1), (2) and (6) of Def. 1.

Theorem 1 *In a* (t,k,n)*-secure KIE,*

$$H(C_{ij}) \geq H(M_{ij}) \quad (i = 0,1,\cdots,\ j = 1,\cdots,n). \tag{1}$$

Now, a lower bound on the required memory size for a user's secret key is shown. Again, this bound can be obtained from Shannon's and from conditions (1), (2) and (6) of Def. 1.

Theorem 2 *In a* (t,k,n)*-secure KIE,*

$$H(U\!K_i) \geq \max_{\{j_0,\cdots,j_k\}\subseteq\{1,\cdots,n\}} H(M_{ij_0},\cdots,M_{ij_k}) \quad (i = 0,1,\cdots). \tag{2}$$

Third, a lower bound on the required memory size for a helper key is shown.

Theorem 3 *In a (t,k,n)-secure KIE,*

$$H(MK_i) \geq \max_{\{j_0,\cdots,j_k\}\subseteq\{1,\cdots,n\}} H(M_{ij_0},\cdots,M_{ij_k}) \quad (i=0,1,\cdots). \tag{3}$$

Proof. From conditions (1) and (3) of Def. 1, we have

$$H(M_{ij}|C_{ij},UK_{i-1},MK_i)=0 \quad (i=0,1,\cdots,\ j=1,\cdots,n).$$

From condition (6) of Def. 1, we also have

$$H(M_{ij}|C_{ij},UK_{i-1})=H(M_{ij}) \quad (i=0,1,\cdots,\ j=1,\cdots,n).$$

Consequently, from these equations, we have Eq. 3. □

Next, a lower bound on the required memory size for the master helper key is shown.

Theorem 4 *In a (t,k,n)-secure KIE,*

$$H(MK^*) \geq \max_{\{i_0,\cdots,i_t\}\subseteq\{0,1,\cdots\}} \sum_{i\in\{i_0,\cdots,i_t\}} \max_{\{j_0,\cdots,j_k\}\subseteq\{1,\cdots,n\}} H(M_{ij_0},\cdots,M_{ij_k}). \tag{4}$$

Proof. From conditions (1) and (6) of Def. 1, we have

$$H(UK_i|C_{ij},UK_{\kappa_1},\cdots,UK_{\kappa_t}) \geq H(M_{ij})$$
$$(i=0,1,\cdots,\ i \notin \forall\{\kappa_1,\cdots,\kappa_t\}\subset\{0,1,\cdots\},\ j=1,\cdots,n).$$

Consequently, we have

$$\begin{aligned} H(UK_i,UK_{\kappa_1},\cdots,UK_{\kappa_t}) &= H(UK_i)+H(UK_{\kappa_1}|UK_i)+H(UK_{\kappa_2}|UK_i,UK_{\kappa_1}) \\ &\quad +\cdots+H(UK_{\kappa_t}|UK_i,UK_{\kappa_1},\cdots,UK_{\kappa_{t-1}}) \\ &\geq \sum_{\ell\in\{i,\kappa_1,\cdots,\kappa_t\}} \max_{\{j_0,\cdots,j_k\}\subseteq\{1,\cdots,n\}} H(M_{\ell j_0},\cdots,M_{\ell j_k}) \\ &\qquad (i=0,1,\cdots,\ i\notin\forall\{\kappa_1,\cdots,\kappa_t\}\subset\{0,1,\cdots\}). \end{aligned}$$

From conditions (3) and (4), we also have

$$H(UK_i,UK_{\kappa_1},\cdots,UK_{\kappa_t}|MK^*,UK_0)=0,$$

and

$$H(UK_i,UK_{\kappa_1},\cdots,UK_{\kappa_t}|UK_0)=\sum_{\ell\in\{i,\kappa_1,\cdots,\kappa_t\}} \max_{\{j_0,\cdots,j_k\}\subseteq\{1,\cdots,n\}} H(M_{\ell j_0},\cdots,M_{\ell j_k})$$
$$(i=1,2,\cdots,\ i\notin\forall\{\kappa_1,\cdots,\kappa_t\}\subset\{1,2,\cdots\}).$$

Hence, we have

$$H(MK^*) \geq \sum_{\ell\in\{i,\kappa_1,\cdots,\kappa_t\}} \max_{\{j_0,\cdots,j_k\}\subseteq\{1,\cdots,n\}} H(M_{\ell j_0},\cdots,M_{\ell j_k})$$
$$(i=1,2,\cdots,\ i\notin\forall\{\kappa_1,\cdots,\kappa_t\}\subset\{1,2,\cdots\}).$$

Therefore, we have

$$H(MK^*) \geq \max_{\{i_0,\cdots,i_t\}\subseteq\{0,1,\cdots\}} \sum_{i\in\{i_0,\cdots,i_t\}} \max_{\{j_0,\cdots,j_k\}\subseteq\{1,\cdots,n\}} H(M_{ij_0},\cdots,M_{ij_k}),$$

which proves the theorem. □

Finally, a lower bound on the required memory size for a sender's key is shown.

Theorem 5 *In a (t,k,n)-secure KIE,*

$$H(E_j) \geq \max_{\{i_0,\cdots,i_t\}\subseteq\{0,1,\cdots\}} H(M_{i_0j},\cdots,M_{i_tj}) \quad (j=1,\cdots,n). \tag{5}$$

Proof. From conditions (1) and (8), we have

$$H(C_{i_0j},\cdots,C_{i_tj}|M_{i_0j},\cdots,M_{i_tj},E_j)=0,$$

and

$$H(M_{i_0j},\cdots,M_{i_tj}|C_{i_0j},\cdots,C_{i_tj}) = H(M_{i_0j},M_{i_1j},\cdots,M_{i_tj})$$
$$(\forall\{i_0,i_1,\cdots,i_t\}\subset\{0,1,\cdots\},\ j=1,\cdots,n).$$

From Shannon's, we have Eq. 5. □

Definition 2 A (t,k,n)-secure KIE is *optimal* if one has equalities in Eq. 1, 2, 3, 4 and 5.

3 Construction

We show a construction of a (t,k,n)-secure KIE which is based on polynomials. In this construction, we assume that the distribution of M_{ij} is uniform for any i,j such that $i=0,1,\cdots$ and $j=1,\cdots,n$, and is also independent of any other variable.

Polynomial Construction. Let q be a prime power, and $GF(q)$ be a finite field with q elements. We assume that the maximum number of key updating that a user is allowed is less than q and a plaintext can be expressed by an element in $GF(q)$. In the initial phase, U generates random polynomials $f(x) := \sum_{i=0}^{k} a_i x^i$ and $mk^*(x,y) := \sum_{i=0}^{k}\sum_{j=0}^{t} b_{ij}x^i y^j$ over $GF(q)$. U distributes $mk^*(x,y), e_1(y) := f(S_1)+mk^*(S_1,y),\cdots,e_n(y) := f(S_n)+mk^*(S_n,y)$ to $H,S_1,\cdots,S_n$, respectively, via a secure channel, assuming that S_i $(1\leq i\leq n)$ are distinct elements in $GF(q)$. Let p_i $(i=0,1,\cdots,\ i<q)$ be public and distinct elements in $GF(q)$. U keeps $uk_0(x) := f(x)+mk^*(x,p_0)$ as the stage 0 user secret key. After distributing the keys, U deletes $mk^*(x,y),e_1(y),\cdots,e_n(y)$ from his memory. For updating the user's key for stage i $(i<q)$, H sends $mk_i(x) := mk^*(x,p_i)-mk^*(x,p_{i-1})$ to U, and U computes $uk_i(x) := uk_{i-1}(x)+mk_i(x)$. U further deletes $uk_{i-1}(x)$ and $mk_i(x)$ from his memory. A sender, $S_j\in\{S_1,\cdots,S_n\}$, encrypts a message $m_{ij}\in GF(q)$ as $c_{ij} := m_{ij}+e_n(p_i)$, and generates a ciphertext c_{ij}. U recovers m_{ij} as $m_{ij}=c_{ij}-uk_i(S_j)$.

Theorem 6 *The above scheme is an optimal (t, k, n)-secure KIE.*

Proof. It is obvious that the above scheme satisfies all of conditions in Def. 1 and has equalities in Eq. 1, 2, 3, 4 and 5. □

Also, we see that the lower bounds in Theorem 1,2,3,4 and 5 are all tight. We note that conditions (5) and (7) are not utilized for the estimation of the bounds. This implies that no additional memory will be necessary in order to satisfy these two conditions.

4 Dynamic and Mutual Key Insulated Encryption

The model of KIE described in section 2 is built for a single receiver, which means that there only exists one receiver for the entire model. We can extend this model to a multiple receiver model and call it, dynamic and mutual key insulated encryption (DMKIE). In DMKIE, each user in a system has a secure device and can both encrypt and decrypt in each of the stages.

4.1 The Model

The model of DMKIE involves n users $U_1, \cdots, U_n$ and each users' trusted devices $H_1, \cdots, H_n$ corresponding to $U_1, \cdots, U_n$, respectively. In the initial phase, a trusted authority or a collaboration of users generates U_i's stage 0 secret key $uk_0^{(i)}$ and U_i's master helper key $mk^{(i),*}$ $(i = 1, \cdots, n)$. These keys are distributed to the corresponding owners of the keys via secure channels. If a trusted authority has been introduced to generated these secrets, this is the point where he deletes $uk_0^{(i)}$ and $mk^{(i),*}$ $(i = 1, \cdots, n)$ from his memory. For U_i's key for stage j updates, H_i sends U_i's stage j helper key, $mk_j^{(i)}$ to U_i which is computed as a function of $mk^{(i),*}$ and j. Then, U_i calculates U_i's stage j user secret key, $uk_j^{(i)}$, as a function of $uk_{j-1}^{(i)}$ and $mk_j^{(i)}$. U_i further deletes $uk_{j-1}^{(i)}$ and $mk_j^{(i)}$ from his memory. When a user, $U_{\ell_0} \in \{U_1, \cdots, U_n\}$ transmits another user $U_{\ell_1} \in \{U_1, \cdots, U_n\} \backslash \{U_{\ell_0}\}$ a message $m_j^{(\ell_0,\ell_1)}$, in stage j, U_{ℓ_0} encrypts $m_j^{(\ell_0,\ell_1)}$ by $uk_j^{(\ell_0)}$ with encryption in stage j performed as a function of $uk_j^{(\ell_0)}$, U_{ℓ_1} and $m_j^{(\ell_0,\ell_1)}$ and generates a ciphertext $c_j^{(\ell_0,\ell_1)}$. Finally, U_{ℓ_1} recovers $m_j^{(\ell_0,\ell_1)}$ as a function of $uk_j^{(\ell_1)}$, U_{ℓ_0} and $m_j^{(\ell_0,\ell_1)}$. We assume that only one message is transmitted between only one pair of users per stage. We assume that there exist at most k malicious users, and at most t user secret keys can be exposed per user.

4.2 Security Definition

The model of DMKIE needs to satisfy the following requirements:(1) U_{ℓ_1} can correctly recover $m_j^{(\ell_0,\ell_1)}$ from $c_j^{(\ell_0,\ell_1)}$ with probability 1. (2) U_{ℓ_0} can correctly create $c_j^{(\ell_0,\ell_1)}$ from $m_j^{(\ell_0,\ell_1)}$ with probability 1. (3) $uk_j^{(i)}$ can correctly be generated from $uk_{j-1}^{(i)}$ and $mk_j^{(i)}$. (4) $mk_j^{(i)}$ can only be generated from $mk^{(i),*}$.

(5) Even if $mk^{(i),*}$ is exposed, none of U_i's secret keys can be computed without also having one of U_i's secret keys. (6) Any coalition of adversaries who may have obtained t exposed U_{ℓ_0}'s secret keys and t exposed U_{ℓ_1}'s secret keys, still cannot gain any more information on $m_j^{(\ell_0,\ell_1)}$ from $c_j^{(\ell_0,\ell_1)}$. (7) Any coalition of adversaries who may have $mk^{(\ell_1),*}$, cannot obtain any information on $m_j^{(\ell_0,\ell_1)}$ from $c_j^{(\ell_0,\ell_1)}$ unless they have one of U_{ℓ_0}'s or U_{ℓ_1}'s secret keys. (8) No ciphertext assist on leaking any information regarding any plaintext. Letting $U\!K_j^{(i)}$ $(i = 1,\cdots,n,\ j = 0,1,\cdots), M\!K^{(i),*}$ $(i = 1,\cdots,n), M\!K_j^{(i)}$ $(i = 1,\cdots,n,\ j = 0,1,\cdots), M_\ell^{(ij)}$ $(i = 1,\cdots,n,\ j = 1,\cdots,n,\ i \neq j,\ \ell = 0,1,\cdots)$ and $C_\ell^{(ij)}$ $(i = 1,\cdots,n,\ j = 1,\cdots,n,\ i \neq j,\ \ell = 0,1,\cdots)$ be random variables induced by $uk_j^{(i)}$ $(i = 1,\cdots,n,\ j = 0,1,\cdots), mk^{(i),*}$ $(i = 1,\cdots,n), mk_j^{(i)}$ $(i = 1,\cdots,n,\ j = 0,1,\cdots), m_\ell^{(ij)}$ $(i = 1,\cdots,n,\ j = 1,\cdots,n,\ i \neq j,\ \ell = 0,1,\cdots)$ and $c_\ell^{(ij)}$ $(i = 1,\cdots,n,\ j = 1,\cdots,n,\ i \neq j,\ \ell = 0,1,\cdots)$, respectively, and now a (t,k,n)-secure DMKIE is formally defined as follows:

Definition 3 A DMKIE in the above model is called a (t,k,n)-*secure DMKIE* if the following equations hold;

(1) $H(M_\ell^{(ij)}|C_\ell^{(ij)}, U\!K_\ell^{(j)}) = 0 \quad (i = 1,\cdots,n,\ j = 1,\cdots,n,\ \ell = 0,1,\cdots)$

(2) $H(C_\ell^{(ij)}|M_\ell^{(ij)}, U\!K_\ell^{(i)}) = 0 \quad (i = 1,\cdots,n,\ j = 1,\cdots,n,\ \ell = 0,1,\cdots)$

(3) $H(U\!K_j^{(i)}|U\!K_{j-1}^{(i)}, M\!K_j^{(i)}) = 0 \quad (i = 1,\cdots,n,\ j = 1,2,\cdots)$

(4) $H(M\!K_j^{(i)}|M\!K^{(i),*}) = 0 \quad (i = 1,\cdots,n,\ j = 1,2,\cdots)$

(5) $H(U\!K_j^{(i)}|M\!K^{(i),*}) = H(U\!K_j^{(i)}) \quad (i = 1,\cdots,n,\ j = 0,1,\cdots)$

(6) $H(M_\ell^{(ij)}|C_\ell^{(ij)}, M\!K^{(\delta_1),*},\cdots,M\!K^{(\delta_k),*}, U\!K_0^{(\delta_1)},\cdots,U\!K_0^{(\delta_k)},$
$U\!K_{\kappa_1}^{(i)},\cdots,U\!K_{\kappa_t}^{(i)}, U\!K_{\omega_1}^{(j)},\cdots,U\!K_{\omega_t}^{(j)}) = H(M_\ell^{(ij)})$
$(i = 1,\cdots,n,\ j = 1,\cdots,n,\ \ell = 0,1,\cdots,\ i,j \notin \forall\{\delta_1,\cdots,\delta_k\} \subset \{1,\cdots,n\},$
$\ell \notin \forall\{\kappa_1,\cdots,\kappa_t\} \subset \{0,1,\cdots\},\ \ell \notin \forall\{\omega_1,\cdots,\omega_t\} \subset \{0,1,\cdots\})$

(7) $H(M_\ell^{(ij)}|M\!K^{(j),*}, C_\ell^{(ij)}, M\!K^{(\delta_1),*},\cdots,M\!K^{(\delta_k),*}, U\!K_0^{(\delta_1)},\cdots,U\!K_0^{(\delta_k)})$
$= H(M_\ell^{(ij)})$
$(i = 1,\cdots,n,\ j = 1,\cdots,n,\ \ell = 0,1,\cdots,\ i,j \notin \forall\{\delta_1,\cdots,\delta_k\} \subset \{1,\cdots,n\})$

(8) $H(M_{\ell_0}^{i_0j_0}, M_{\ell_1}^{i_1j_1},\cdots|C_\beta^{\delta\gamma}) = H(M_{\ell_0}^{i_0j_0}, M_{\ell_1}^{i_1j_1},\cdots)$
$(\forall\{i_0,i_1,\cdots\} \subseteq \{1,\cdots n\},\ \forall\{j_0,j_1,\cdots\} \subseteq \{1,\cdots,n\},$
$\forall\{\ell_0,\ell_1,\cdots\} \subseteq \{1,2,\cdots\}, \delta = 1,\cdots,n,\ \gamma = 1,\cdots,n,\ \beta = 0,1,\cdots)$

4.3 Construction

In this subsection, we show a construction of DMKIE based on polynomials.

Polynomial Construction. Let q be a prime power, and $GF(q)$ be a finite field with q elements. We assume that the maximum number of key updating that a user is allowed is less than q times, and a plaintext can be expressed as an element in $GF(q)$. We also assume that U_i $(i = 1, \cdots, n)$ are expressed as distinct elements in $GF(q)$. Let p_i $(i = 0, 1, \cdots, \ i < q)$ be public and are distinct elements in $GF(q)$. In the initial phase, either a trusted authority or a collaboration of users generates U_i's stage 0 secret key as $uk_0^{(i)}(x) := f(x, U_i) + \sum_{j=0}^{t} g_j(x, U_i){p_0}^j$ as well as his master helper key as $mk^{(i),*}(x, z) := \sum_{j=0}^{t} g_j(x, U_i)z^j$ $(i = 1, \cdots, n)$, where $f(x, y)$ and $g_j(x, y)$ $(j = 0, \cdots, t)$ are random symmetric polynomials over $GF(q)$ such that the maximum degrees of x and y are at most k. (A method of distributed generation of symmetric polynomials is shown in [18].) Now, this is when the trusted authority deletes $uk_0^{(i)}(x)$ and $mk^{(i),*}(x, z)$ $(i = 1, \cdots, n)$ from his memory after generating these secrets. For updating U_i's key for stage j, H_i sends U_i, U_i's stage j helper key as $mk_j^{(i)}(x) := mk^{(i),*}(x, p_j) - mk^{(i),*}(x, p_{j-1})$. Then, U_i calculates U_i's stage j user secret key as $uk_j^{(i)}(x) := uk_{j-1}^{(i)}(x) + mk_j^{(i)}(x)$. Moreover, U_i deletes $uk_{j-1}^{(i)}(x)$ and $mk_j^{(i)}(x)$ from his memory. When a user, $U_{\ell_0} \in \{U_1, \cdots, U_n\}$, transmits another user $U_{\ell_1} \in \{U_1, \cdots, U_n\} \backslash \{U_{\ell_0}\}$ a message $m_j^{(\ell_0, \ell_1)}$ in stage j, U_{ℓ_0} will now encrypt $m_j^{(\ell_0, \ell_1)}$ as $c_j^{(\ell_0, \ell_1)} := m_j^{(\ell_0, \ell_1)} + uk_j^{(\ell_0)}(U_{\ell_1})$. Finally, U_{ℓ_1} recovers $m_j^{(\ell_0, \ell_1)}$ as $m_j^{(\ell_0, \ell_1)} = c_j^{(\ell_0, \ell_1)} - uk_j^{(\ell_1)}(U_{\ell_0})$.

Theorem 7 *The above scheme is a (t, k, n)-secure DMKIE.*

5 DMKIE from KPS and BES

In this section, we show how closely related DMKIE is to *key predistribution schemes* (KPS) [4,14,5,17,12] and *broadcast encryption schemes* (BES) [11,6,17]. More specifically, we show that a DMKIE can be constructed from either KPS or BES.

In KPS, either a trusted authority or a collaboration of users, generates the secret information which is transmitted to a set of users. Member of privileged subset P of set of users can compute a common key, k_p, which is specific to them. While, no coalition F (of forbidden subset of users) is able to recover the information of the key k_p. In BES, a trusted authority or a collaboration of users, generates and distributes the secret information to set of users and broadcasts ciphertext b_p over a network. The secret information is generated so that each member of a particular subset P of the users can decrypt b_p, but no coalition F (of forbidden subset of users) is able to recover the information of the plaintext m_p of b_p.

5.1 DMKIE from KPS

The Model of KPS. Let $\{U_1, \cdots, U_n\}$ be a set of users. In KPS, trusted authority or a collaboration of users generates and distributes secret information to each user. The information given to user U_i is denoted as $uk_{\mathsf{KPS}_j}^{(i)}$ and is distributed via a secure channel, where j is an index in case there exist multiple KPSs for the same user. Such information enables various *privileged subsets* to compute their common keys. Let $2^{\mathcal{U}}$ denote the set of all subsets of users, $\mathcal{P} \subseteq 2^{\mathcal{U}}$, the set of all privileged subsets to whom TA distributes the keys, and $\mathcal{F} \subseteq 2^{\mathcal{U}}$, the set of all possible coalitions (called *forbidden subsets*) against whose key that is to remain secure. We assume that $\mathcal{P}$ and $\mathcal{F}$ are common to all KPSs for the same users. Once the secret information is distributed, each user U_i in a privileged set P should be able to compute the key $k_{j\,P}$ associated with P. While, no forbidden set $F \in \mathcal{F}$ disjoint from P, should be able to compute $k_{j\,P}$. Let $K_{j\,P}$ and $U\!K^{(i)}_{\mathsf{KPS}_j}$ $(i = 1, \cdots, n)$ be random variables induced by $k_{j\,P}$ and $uk^{(i)}_{\mathsf{KPS}_j}$ $(i = 1, \cdots, n)$, respectively.

Definition 4 $(uk^{(1)}_{\mathsf{KPS}_j}, \cdots, uk^{(n)}_{\mathsf{KPS}_j})$ is a (τ, k, n)-secure KPS if

$$\begin{aligned} &H(K_{j\,P}|U\!K^{(i)}_{\mathsf{KPS}_j}) = 0 \quad (\forall U_i \in P,\ \forall P \subseteq \mathcal{P},\ |P| = \tau),\\ &H(K_{j\,P}|U\!K^{(i_1)}_{\mathsf{KPS}_j}, \cdots, U\!K^{(i_k)}_{\mathsf{KPS}_j}) = H(K_{j\,P})\\ &\quad (\forall P \subseteq \mathcal{P},\ \forall \{U_{i_1}, \cdots, U_{i_k}\} \subseteq F,\ \forall F \in \mathcal{F},\ P \cap F = \phi). \end{aligned}$$

It is clear that there exist an operation $\diamond$, such that for secret information in a (τ, k, n)-secure KPS, $\diamond$ satisfy *linear* property, namely, for any integers a, b and any pair of (τ, k, n)-secure KPSs over the finite field; $(uk^{(1)}_{\mathsf{KPS}_{j_0}}, \cdots, uk^{(n)}_{\mathsf{KPS}_{j_0}})$ and $(uk^{(1)}_{\mathsf{KPS}_{j_1}}, \cdots, uk^{(n)}_{\mathsf{KPS}_{j_1}})$, we have another (τ, k, n)-secure KPS over the finite field, $(uk^{(1)}_{\mathsf{KPS}_{j_2}}, \cdots, uk^{(n)}_{\mathsf{KPS}_{j_2}}) := (a \cdot uk^{(1)}_{\mathsf{KPS}_{j_0}} \diamond b \cdot uk^{(1)}_{\mathsf{KPS}_{j_1}}, \cdots, a \cdot uk^{(n)}_{\mathsf{KPS}_{j_0}} \diamond b \cdot uk^{(n)}_{\mathsf{KPS}_{j_1}})$, such that $k_{j_2\,P} = a\ k_{j_0\,P} + b\ k_{j_1\,P}$, where $x \cdot uk$ denotes $\diamond_{i=1}^{x} uk$ for any integer x and secret information uk.

A Construction of DMKIE from KPS. Here, we demonstrate a construction of DMKIE using KPS.

<u>*KPS-based Construction.*</u> Let $\mathcal{P}$ be $\{X | X \in 2^{\mathcal{U}},\ |X| = 2\}$. Let $(uk^{(1)}_{\mathsf{KPS}_\ell}, \cdots, uk^{(n)}_{\mathsf{KPS}_\ell})$ $(\ell = 0, \cdots, t+1)$ be independent $(2, k, n)$-secure KPSs over a finite field $GF(q)$ with q elements, where q is a prime. It is required that no information on $\diamond_{\ell=1}^{t+1} {j_0}^{\ell-1} \cdot uk^{(i)}_{\mathsf{KPS}_\ell}$ can be obtained from $\{\diamond_{\ell=1}^{t+1} {j_1}^{\ell-1} \cdot uk^{(i)}_{\mathsf{KPS}_\ell}, \cdots, \diamond_{\ell=1}^{t+1} {j_t}^{\ell-1} \cdot uk^{(i)}_{\mathsf{KPS}_\ell}\}$, and that no information on $uk^{(i)}_{\mathsf{KPS}_0} \diamond \diamond_{\ell=1}^{t+1} {j_0}^{\ell-1} \cdot uk^{(i)}_{\mathsf{KPS}_\ell}$ can be obtained from $uk^{(i)}_{\mathsf{KPS}_1}, \cdots, uk^{(i)}_{\mathsf{KPS}_{t+1}}$, for any $i \in \{1, \cdots, n\}$ and distinct $j_0, \cdots, j_t \in GF(q)$. KPSs in [4,14,5,11] satisfy this requirement. We assume that the maximum number of key updating that a user is allowed is less than q. Let U_i's stage 0 secret key $uk^{(i)}_0 := uk^{(i)}_{\mathsf{KPS}_0} \diamond uk^{(i)}_{\mathsf{KPS}_1}$ and U_i's master helper key $mk^{(i),*} := \{uk^{(i)}_{\mathsf{KPS}_1}, \cdots, uk^{(i)}_{\mathsf{KPS}_{t+1}}\}$ $(i = 1, \cdots, n)$. For updating U_i's key for stage j, H_i sends to U_i, U_i's stage j helper key $mk^{(i)}_j$, such that $\diamond_{\ell=1}^{t+1} j^{\ell-1} \cdot uk^{(i)}_{\mathsf{KPS}_\ell} = mk^{(i)}_j \diamond$

$\diamond_{\ell=1}^{t+1}(j-1)^{\ell-1}\cdot uk_{\mathsf{KPS}_\ell}^{(i)}$. Then, U_i calculates U_i's stage j user secret key $uk_j^{(i)} := uk_{j-1}^{(i)} \diamond mk_j^{(i)}$. Moreover, U_i deletes $uk_{j-1}^{(i)}$ and $mk_j^{(i)}$ from his memory. When a user, $U_{\ell_0} \in \{U_1,\cdots,U_n\}$, transmits another user $U_{\ell_1} \in \{U_1,\cdots,U_n\}\backslash\{U_{\ell_0}\}$, a message $m_j^{(\ell_0,\ell_1)}$ in stage j, then, U_{ℓ_0} encrypts $m_j^{(\ell_0,\ell_1)}$ as $c_j^{(\ell_0,\ell_1)} := m_j^{(\ell_0,\ell_1)} + k_{j\{U_{\ell_0},U_{\ell_1}\}}$, such that the size of $m_j^{(\ell_0,\ell_1)}$ is equivalent or less than that of $k_{j\{U_{\ell_0},U_{\ell_1}\}}$, where $k_{j\{U_{\ell_0},U_{\ell_1}\}}$ is the shared key between U_{ℓ_0} and U_{ℓ_1} by KPS $(uk_{\mathsf{KPS}_0}^{(1)} \diamond \diamond_{\ell=1}^{t+1}(j-1)^{\ell-1}\cdot uk_{\mathsf{KPS}_\ell}^{(1)},\cdots,uk_{\mathsf{KPS}_0}^{(n)} \diamond \diamond_{\ell=1}^{t+1}(j-1)^{\ell-1}\cdot uk_{\mathsf{KPS}_\ell}^{(n)})$. Finally, U_{ℓ_1} recovers $m_j^{(\ell_0,\ell_1)}$ as $m_j^{(\ell_0,\ell_1)} = c_j^{(\ell_0,\ell_1)} - k_{j\{U_{\ell_0},U_{\ell_1}\}}$.

Theorem 8 *The above scheme is a (t,k,n)-secure DMKIE.*

5.2 DMKIE from BES

The Model of BES. Let $\{U_1,\cdots,U_n\}$ be a set of users. In a BES, we assume that the network is a *broadcast channel*, i.e., it is insecure, and that any information transmitted by a trusted authority (TA) will be received by every user. In the set-up stage, TA generates and distributes secret information $uk_{\mathsf{BES}}^{(i)}$ to each user U_i via a secure channel. Later, the TA broadcasts a message m_P to privileged subset P. Which particular privileged subset P is selected, is in general, not known ahead of time. $\mathcal{P} \subseteq 2^{\mathcal{U}}$ denotes the set of all privileged subsets to which the TA might want to broadcast a message, and $\mathcal{F} \in 2^{\mathcal{U}}$, the set of all possible coalitions (forbidden subsets) against which a broadcast is to remain secure. Suppose that, TA wants to broadcast a message to a given privileged set $P \in \mathcal{P}$ at time not specified. In that case, TA broadcast b_P which is computed as a function of m_P and all $uk_{\mathsf{BES}}^{(i)}$ such that $U_i \in P$. Once b_P has been broadcast, each user $U_i \in P$ should be able to decrypt b_P and obtain m_P. On the other hand, no forbidden set $F \in \mathcal{F}$ disjoint from P should be able to compute m_P. Let M_P, B_P and $U\!K_{\mathsf{BES}}^{(i)}$ $(i=1,\cdots,n)$ be random variables induced by m_P, b_P and $uk_{\mathsf{KPS}}^{(i)}$ $(i=1,\cdots,n)$, respectively.

Definition 5 $(U\!K_{\mathsf{BES}}^{(1)},\cdots,U\!K_{\mathsf{BES}}^{(n)})$ is a (τ,k,n)-secure BES if

$$\begin{aligned}
&H(M_P|U\!K_{\mathsf{BES}}^{(1)},\cdots,U\!K_{\mathsf{BES}}^{(n)}) = H(M_P)\\
&H(M_P|B_P,U\!K_{\mathsf{BES}}^{(i)}) = 0 \quad (\forall U_i \in P,\ \forall P \subseteq \mathcal{P},\ |P| = \tau),\\
&H(M_P|B_P,U\!K_{\mathsf{BES}}^{(i_1)},\cdots,U\!K_{\mathsf{BES}}^{(i_k)}) = H(K_P)\\
&\quad (\forall P \subseteq \mathcal{P},\ \forall\{U_{i_1},\cdots,U_{i_k}\} \subseteq F,\ \forall F \in \mathcal{F},\ P \cap F = \phi).
\end{aligned}$$

A Construction of DMKIE from BES. As shown in [13], we can construct a (τ,k,n)-secure KPS from a (τ,k,n)-secure BES. Therefore, a $(2,k,n)$-secure KPS can be constructed from a $(2,k,n)$-secure BES. Consequently, a (t,k,n)-secure DMKIE can be constructed from a $(2,k,n)$-secure BES.

References

1. M. Abdalla and L. Reyzin, "A New Forward-Secure Digital Signature Scheme," Proc. of Asiacrypt 2000, LNCS 1976, Springer-Verlag, pp. 116-129,2000.
2. M. Bellare and A. Palacio, "Protecting against Key Exposure: Strongly Key-Insulated Encryption with Optimal Threshold," available at `http://eprint.iacr.org/2002/064/` .
3. M. Bellare and S.K. Miner, "A Forward-Secure Digital Signature Scheme," Proc. of CRYPTO'99, LNCS 1666, Springer-Verlag, pp. 431-448, 1999.
4. R. Blom, "Non-public Key Distribution," Proc. of CRYPTO'82, Plenum Press, pp.231-236, 1983.
5. C. Blundo, A. DeSantis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung, "Perfectly Secure Key Distribution for Dynamic Conferences," Proc. of CRYPTO'92, LNCS 740, Springer-Verlag, pp.471-486, 1993.
6. C. Blundo, L.A. Frota Mattos and D.R. Stinson, "Trade-offs between Communication and Strage in Unconditionally Secure Schemes for Broadcast Encryption and Interactive Key Distribution," Proc. of CRYPTO'96, LNCS 1109, Springer-Verlag, pp.387-400, 1996.
7. D. Boneh and M. Franklin, "Identity-Based Encryption from the Weil Pairing," Proc. of CRYPTO 2001, LNCS 2139, Springer-Verlag, pp.213-229, 2001.
8. R. Canetti and S. Goldwasser, "An Efficient Threshold Public-Key Cryptosystem secure against adaptive chosen-ciphertext attack," Proc. of Eurocrypt'99, LNCS 1592, Springer-Verlag, pp.90-106, 1999.
9. C. Cocks, "An Identity Based Encryption Scheme Based on Quadratic Residues," Proc. of IMA Int. Conf. 2001, Coding and Cryptography, LNCS 2260, Springer-Verlag, pp. 360-363, 2001.
10. Y. Dodis, J. Katz, S. Xu and M. Yung, "Key-Insulated Public Key Cryptosystems," Proc. of Eurocrypt 2002, LNCS 2332, Springer-Verlag, pp.65-82, 2002.
11. A. Fiat and M. Naor, "Broadcast Encryption," Proc. of CRYPTO '93, LNCS 773, Springer-Verlag, pp.480-491, 1994.
12. G. Hanaoka, T. Nishioka, Y. Zheng and H. Imai, "An Efficient Hierarchical Identity-based Key-Sharing Method Resistant against Collusion-Attacks," Proc. of Asiacrypt'99, LNCS 1716, Springer-Verlag, pp.348-362, 1999.
13. K. Kurosawa, T. Yoshida, Y. Desmedt and M. Burmester, "Some Bounds and a Construction for Secure Broadcast Encryption," Proc. of Asiacrypt'98, LNCS 1514, Springer-Verlag, pp.420-433, 1998.
14. T. Matsumoto and H. Imai, "On the KEY PREDISTRIBUTION SYSTEM: A Practical Solution to the Key Distribution Problem," Proc. of CRYPTO'87, LNCS 293, Springer-Verlag, pp.185-193, 1987.
15. A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," Proc. of CRYPTO'84, LNCS 196, Springer-Verlag, pp.47-53, 1985.
16. V. Shoup and R. Gennaro, "Securing Threshold Cryptosystems Against Chosen-Ciphertext Attack," Proc. of Eurocrypt'99, LNCS 1403, Springer-Verlag, pp.1-16, 1998.
17. D.R. Stinson, "On Some Methods for Unconditionally Secure Key distribution and broadcast encryption," Designs, Codes and Cryptography, 12, pp.215-243, 1997.
18. D.R. Stinson and R. Wei, "Unconditionally Secure Proactive Secret Sharing Scheme with Combinatorial Structures," Proc. SAC'99, LNCS 1758, Springer-Verlag, pp.200-214, 1999.

Anonymous Fingerprinting as Secure as the Bilinear Diffie-Hellman Assumption

Myungsun Kim, Jongseong Kim, and Kwangjo Kim

International Research center for Information Security (IRIS)
Information and Communications University (ICU)
58-4, Hwaamdong, Yuseong-gu, Daejeon, 305-732, Korea
{ms.kim,jskim224,kkj}@icu.ac.kr

Abstract. The illegal copying and redistribution of digitally-stored information is a crucial problem to distributors who electronically sell digital data. Fingerprinting provides a means which a copyright owner can trace illegal redistributors of electronic information. Various fingerprinting schemes have appeared as techniques for copyright protection from symmetric fingerprinting by Boneh and Shaw [3], asymmetric fingerprinting by Pfitzmann and Schunter [14], and anonymous fingerprinting by Pfitzmann and Waidner [15]. In most of previous schemes, the computational capability of clients has been assumed to roughly be equal to each other and even to their servers. In particular, the key size of known algorithms for fingerprinting schemes keeps back from their practical implementation. In this paper, we propose a scheme for anonymous fingerprinting based on the bilinear Diffie-Hellman problem and prove its security. Our scheme exhibits all computations are performed more efficiently than previous schemes and the key size is quite reasonable for practical use.

Keywords: Anonymous, asymmetric, and symmetric fingerprinting, Bilinear Diffie-Hellman problem, Intellectual property protection, Security reduction

1 Introduction

According to the progress of computer networks and development of the Internet, protection of digitally-stored information property has become a crucial problem to be solved. A lot of research work has been invested into the design of methods that technically support the copyright protection of digital data. One class of such methods consists of techniques called *fingerprinting schemes*. The other class of such methods is called *watermarking schemes*. Watermarking is clearly one of the reasonable alternatives to solve several problems such as violation of ownership and illegal distribution of the copy. It enables the owner of digital property to embed some information in the digital contents and to extract it. On the other hand, fingerprinting allows a buyer to embed the information related to himself, and enables a merchant to trace the buyer from the illegally redistributed copy.

In general, fingerprinting schemes are classified into two different classes called *symmetric* fingerprinting schemes [2,3,17] and *asymmetric* fingerprinting schemes [14,18]. While in symmetric schemes the merchant fingerprints the data item, asymmetric schemes achieve this in an interactive protocol between the buyer and the merchant where the buyer also embeds his own secret. At the end of the protocol only the buyer knows the fingerprinted data item. The advantage of the asymmetric schemes over the symmetric schemes is for the merchant to obtain a proof of treachery that convinces any honest third party.

The two aforementioned classes of fingerprinting schemes do not preserve privacy because buyers are required to identify themselves to the merchant for the purpose of fingerprinting. Purchasing digital items, especially in open networks, reveals information about the buyer' shopping behavior. Such buyer-profiles are very appealing to commercial misuse. Thus it is desirable for buyers to be capable of purchasing fingerprinted digital items anonymously and remain anonymous as long as they do not distribute the digital contents illegally. To solve this problem, *anonymous asymmetric* fingerprinting schemes were first proposed by Pfitzmann and Waidner [15]. Since then, various anonymous fingerprinting schemes have been proposed in [10,11,8,9,13,5]. The construction in [10,11,8] is based on general two-party computation. The scheme [9] uses oblivious transfer protocols. Later, Kuribayashi and Tanaka [13] focused on improving enciphering rate. Another approach for constructing anonymous fingerprinting schemes based on group signatures was suggested by Camenisch in [5].

However, most of the previous fingerprinting schemes, especially anonymous cases, have not taken into account the computational capability of buyers. From the practical point of view, the key size of known algorithms for previous fingerprinting schemes keeps back from their practical implementation. In this paper, we propose a scheme for anonymous fingerprinting based on the bilinear Diffie-Hellman problem and analyze its security emphasizing that all computations can be performed efficiently and quite reasonable with respect to the key size.

The outline of this paper is as follows: Section 2 gives a brief introduction of fingerprinting schemes and primitives adopted in this paper. Section 3 contains several notations and formal statements for our definition of security. We proceed in Section 4 by presenting our proposal. We then discuss the security of the proposed method in Section 5 and present the brief comparison with previous schemes in Section 6. Finally, we make concluding remarks in Section 7.

2 Preliminaries

In this section, we introduce some basic techniques used in our scheme. First we review various fingerprinting techniques. Next we state briefly the bilinear Diffie-Hellman problem exploited in the conventional scheme in [1,6].

2.1 Fingerprinting

Digital contents such as image, music, and movie are easily copied without any degradation. Fingerprinting is a cryptographic scheme for the copyright protection of digital contents assisted by a watermarking technique. And the scheme can deter people from executing illegal redistribution of digital contents by making it possible for the merchant to identify the original buyer of the redistributed copy, where we call her a traitor. The fingerprinting schemes can be classified into the following three classes.

Symmetric. The operation to embed a fingerprint is performed only by a merchant. Therefore, the merchant cannot convince any third party of the traitor's treachery even if the merchant has detected the identity of a traitor in the content.

Asymmetric. Fingerprinting is an interactive protocol between a buyer and a merchant. After the purchase, only the buyer obtains the copy with a fingerprint. If the merchant has found the illegally distributed copy somewhere, he can identify the traitor and prove to the third party.

Anonymous (asymmetric). A buyer can purchase a fingerprinted content without revealing his identity to a merchant, however the merchant can identify the traitor when he finds the illegally distributed copy. It also retains the asymmetric property.

Most fingerprinting schemes have a collusion problem. Suppose that digital contents are distributed with different fingerprints. If a collusion group who has obtained those contents compares fingerprints of their contents, they easily capture all fingerprints from their contents. Therefore the collusion group can remove original fingerprints, interpolate gaps, and resell the digital contents without worrying about being traced. This collusion problem was first studied by Blakley *et al.* [2] and practical solution against collusion was dealt with by Boneh and Shaw [3].

2.2 Bilinear Diffie-Hellman Problem

We can make use of any bilinear map on an elliptic curve to construct a group $\mathbb{G}$ in which the computational Diffie-Hellman (C-DH) problem is intractable, but the decisional Diffie-Hellman (D-DH) problem is tractable [1,4].

Let E be an elliptic curve over a base field K and let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of order m for some large prime m. Our scheme makes use of a *bilinear* map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ between these two groups. The bilinear map must satisfy the following properties.

i. *Bilinearity*: For all $P, Q, R \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_m^*$, $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ or $\hat{e}(P + Q, R) = \hat{e}(P, R) \cdot \hat{e}(Q, R)$ and $\hat{e}(P, Q + R) = \hat{e}(P, Q) \cdot \hat{e}(P, R)$.
ii. *Non-degeneracy*: If $\hat{e}(P, Q) = 1$ for all $Q \in \mathbb{G}_1$, then $P = \mathcal{O}$, where $\mathcal{O}$ is a point at infinity.
iii. *Computability*: There is an efficient algorithm to compute $\hat{e}(P, Q)$ for any $P, Q \in \mathbb{G}_1$.

Since the D-DH problem in $\mathbb{G}_1$ is easy, we cannot use the D-DH problem to build cryptosystems in the group $\mathbb{G}_1$. Instead, the security of our protocol is based on a variant of the C-DH problem called the bilinear Diffie-Hellman (B-DH) problem.

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order m and let P be a generator of $\mathbb{G}_1$. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map.

Definition 1. *The* B-DH problem *in* $(\mathbb{G}_1, \mathbb{G}_2, \hat{e})$ *is the following: given* (P, aP, bP, cP) *for some* $a, b, c \in \mathbb{Z}_m^*$, *compute* $v \in \mathbb{G}_2$ *such that* $v = \hat{e}(P, P)^{abc}$.

Definition 2. *A randomized algorithm* $\mathcal{IG}$ *is a* B-DH parameter generator *if*

1. *$\mathcal{IG}$ takes a security parameter $0 < k \in \mathbb{Z}$,*
2. *$\mathcal{IG}$ runs in polynomial time in k, and*
3. *$\mathcal{IG}$ outputs the description of two groups $\mathbb{G}_1, \mathbb{G}_2$ and the description of a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.*

We require that the groups have the same prime order $m = |\mathbb{G}_1| = |\mathbb{G}_2|$. We denote the output of $\mathcal{IG}$ by $\mathcal{IG}(1^k)$. A concrete example of the B-DH parameter generator is given in [1].

3 Definitions of Security

In this section, we present some definitions that should be satisfied by a fingerprinting scheme and its security.

3.1 Bilinear Diffie-Hellman Assumption

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of prime order m and let P be a generator of $\mathbb{G}_1$. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be a bilinear map.

Definition 3. *An algorithm* $\mathcal{A}$ *has an* advantage $\mathsf{Adv}^{\mathsf{B\text{-}DH}}(\mathcal{A}) = \epsilon$ *in solving* B-DH *in* $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle$ *if*

$$\mathsf{Adv}^{\mathsf{B\text{-}DH}}(\mathcal{A}) \triangleq \Pr\left[\mathcal{A}(P, aP, bP, cP) = \hat{e}(P, P)^{abc}\right] \geq \epsilon,$$

where the probability is over the random choice of $\langle a, b, c \rangle \in \mathbb{Z}_m^*$, *the random choice of* $P \in \mathbb{G}_1^*$, *and the random bits of* $\mathcal{A}$.

The security of our fingerprinting scheme is intrinsically based on the intractability of the B-DH problem. We formally describe this assumption, called the bilinear Diffie-Hellman intractability assumption (B-DHIA).

A (τ, ϵ)-B-DH-attacker for the groups is a probabilistic polynomial time(PPT) algorithm $\mathcal{A}$ running in time τ that given a B-DH parameter generator $\mathcal{IG}$ stated in Section 2 solves the B-DH problem if for a sufficiently large k:

$$\Pr\left[\mathcal{A}(\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, aP, bP, cP) = \hat{e}(P, P)^{abc} \;\middle|\; \begin{array}{l} \langle \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle \leftarrow \mathcal{IG}(1^k); \\ P \leftarrow \mathbb{G}_1^*; \\ \langle a, b, c \rangle \leftarrow \mathbb{Z}_m^* \end{array}\right] \geq \epsilon.$$

We denote this probability by $\mathsf{Succ}_{\mathcal{IG}}^{\mathsf{B\text{-}DH}}(\mathcal{A})$.

Definition 4 (B-DHIA). *Given a* B-DH *parameter generator* $\mathcal{IG}$ *the* B-DH *problem is* (τ, ϵ)-intractable *if there is no* (τ, ϵ)*-attacker* $\mathcal{A}$ *for the groups.*

3.2 Model of Anonymous Fingerprinting

A model given in [5] was focused on fingerprinting using group signatures. We describe the more general model of anonymous fingerprinting schemes.

Definition 5 (AAF protocol). *An anonymous (asymmetric) fingerprinting* (AAF) *protocol* $\mathcal{P}_{\mathsf{fing}} = \{\mathsf{FKG}_R, \mathsf{FReg}_{RB}, \mathsf{FAuth}_{MB}, \mathsf{FFing}_{MB}, \mathsf{FIden}_{MR}\}$ *involving a buyer B, a merchant M, and a registration authority R is defined by the followings:*

- FKG_R*: A PPT algorithm for R. Invoking the* B-DH *parameter generator* $\mathcal{IG}$*, it outputs R's secret key and the corresponding public key, which is published authentically.*
- FReg_{RB}*: A probabilistic two-party protocol between B and R. B registers at R and at the end each party obtains a registration record. B outputs his anonymous public-key and obtains certificates on pseudonym pairs.*
- FAuth_{MB}*: A probabilistic two-party protocol between B and M. B authenticates himself to M using the certificate from the sub-protocol* FReg_{RB}*.*
- FFing_{MB}*: A probabilistic two-party protocol between B and M. B buys the data item from M and jointly fingerprints it with him. The output to M is a purchase record and the main output to B is the fingerprinted data item.*
- FIden_{MR}*: A probabilistic two-party protocol between R and M. If M finds an illegally redistributed copy, he extracts some information from this copy. The output to M is a proof which also contains the description of the corresponding data item and the real identity of a traitor. R examines the proof received from M by using the corresponding public information and makes a decision.*

Now we can define the security of the AAF scheme. The definition allows the security of the proposed protocol to be reduced to that of the underlying hard problem.

Definition 6. *A protocol* $\mathcal{P}_{\mathsf{fing}} = \{\mathsf{FKG}_R, \mathsf{FReg}_{RB}, \mathsf{FAuth}_{MB}, \mathsf{FFing}_{MB}, \mathsf{FIden}_{MR}\}$ *is a* secure AAF *protocol if:*

1. ***Correctness:*** *All sub-protocols should terminate successfully whenever all players B, M, and R are honest.*
2. ***Registration security:*** *Without compromising the private key x_B of B, the registration protocol* FReg_{RB} *provides authentication to B.*
3. ***Anonymity:*** *Without obtaining a particular and an illegally redistributed copy, M cannot identify B through* FAuth_{MB} *and* FFing_{MB}*.*

A (t, ε)-AAF-**breaker** for $\mathcal{P}_{\mathsf{fing}}$ is a PPT Turing machine Δ running in time t that satisfies three conditions of Definition 6 at least with probability $\varepsilon = \mathsf{Succ}^{\mathsf{AAF}}_{\mathcal{P}}(\Delta)$. Then the fingerprinting scheme is (t, ε)-AAF-**secure** if there is no (t, ε)-AAF-breaker Δ.

4 The Protocol

In this section, we propose a secure fingerprinting scheme which provides anonymity and overcomes the drawbacks from previous schemes. In Domingo's schemes [10,11], the registration protocol is a 4-pass and his schemes require many exponential operations. Our scheme is a 3-pass registration protocol and requires one addition, one scalar multiplication, and one pairing operation over an elliptic curve under the assumption that pre-computations are possible. The identification protocol in our scheme preserves the same advantage.

Our AAF protocol $\mathcal{P}_{\mathsf{fing}}^{\mathsf{B\text{-}DH}}$consists of three sub procedures: registration, fingerprinting, and identification. The registration procedure involves the key generation algorithm and the fingerprinting procedure may be divided into buyer's authentication process and fingerprinting process. Our scheme is constructed as follows:

4.1 Registration Procedure

R invokes the key generation algorithm FKG_R at first. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic groups of order m for some large prime m, P be an arbitrary generator of $\mathbb{G}_1$, and $\hat{e}$ be a bilinear map such that

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2.$$

Assume that both B and R have the B-DH public-key pairs as in [1]. R uses its secret key to issue certificates which can be verified using R's public key. The public keys of R and all buyers are supposed to be known and certified. The buyer's secret key is $x_B = s_1 s_2 s_3 \in \mathbb{Z}_m^*$ and his public key is $y_B = \hat{e}(x_B P, P) = \hat{e}(P, P)^{s_1 s_2 s_3} \in \mathbb{G}_2$.

Protocol [registration] – FReg_{RB}.

1. R chooses a secret random $x_R \in \mathbb{Z}_m^*$ and sends $T_R = x_R P$ to B.
2. B uses secret keys s_1, s_2, and s_3 in $\mathbb{Z}_m^*$ and computes X and Y such that
$$X = s_1 s_2 P \quad \text{and} \quad Y = s_1 s_2 s_3 P + T_R.$$
B convinces R in zero-knowledge of possession of s_1, s_2 and s_3. Note that Y plays a role of anonymous public-key of B.
3. R checks that $\hat{e}(Y, P) = y_B \cdot \hat{e}(P, T_R)$. If valid, R computes $T = \hat{e}(X, T_R)$. Otherwise terminates the protocol. R returns to B the certificates $\mathsf{Cert}(T)$, $\mathsf{Cert}(Y \| x_R)$, and x_R. The certificates issued by R state the correctness of T and Y.
4. On receiving certificates, B verifies that $T = \hat{e}(X, T_R)$. He views (Y, T) as a pseudonym pair and keeps it safely.

The registration protocol works as shown in Figure 1.

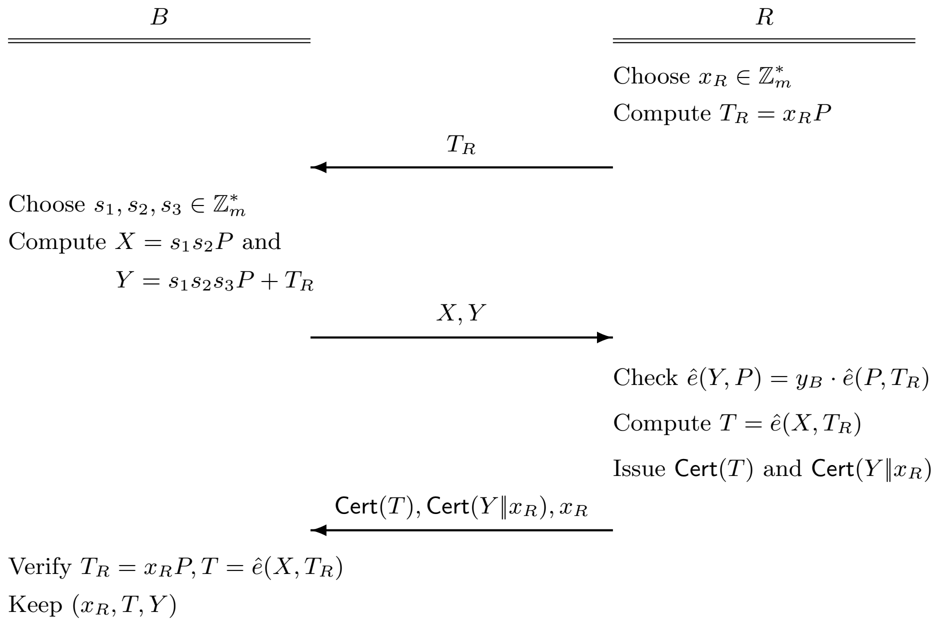

B
R
Choose $x_R \in \mathbb{Z}_m^*$
Compute $T_R = x_R P$
T_R
Choose $s_1, s_2, s_3 \in \mathbb{Z}_m^*$
Compute $X = s_1 s_2 P$ and
$Y = s_1 s_2 s_3 P + T_R$
X, Y
Check $\hat{e}(Y, P) = y_B \cdot \hat{e}(P, T_R)$
Compute $T = \hat{e}(X, T_R)$
Issue $\mathsf{Cert}(T)$ and $\mathsf{Cert}(Y \| x_R)$
$\mathsf{Cert}(T), \mathsf{Cert}(Y \| x_R), x_R$
Verify $T_R = x_R P, T = \hat{e}(X, T_R)$
Keep (x_R, T, Y)

Fig. 1. The registration protocol FReg_{RB}

4.2 Fingerprinting Procedure

From the conceptual point of view, fingerprinting is similar to secure contract signing in some respects. One can capture such a similarity from the following fingerprinting protocol. Assume that the B-DH signature given in [6] or its variants may be used here. If possible, we call the signing algorithm $\mathsf{Sign}(\cdot, \ldots, \cdot)$ and the verifying algorithm $\mathsf{Verify}(\cdot, \cdot, \cdot)$. Assume that a variant of a secure multiparty computation given in [7] may be constructed under the B-DH assumption.

Protocol [fingerprinting] – (FAuth_{MB}, FFing_{MB}).

1. B sends $Y, [T, \mathsf{Cert}(T)]$, and text to M, where text is a string identifying his purchase. B outputs a B-DH signature sig on text with the secret key (s_1, s_2, s_3, x_R). The signature sig is not sent to M.
2. M verifies the certificate $\mathsf{Cert}(T)$ on T and stores $[T, \mathsf{Cert}(T)]$ as his purchase record.
3. B and M initiate a secure two-party computation assumed as above. M's inputs are T, Y, text, and item, where item denotes the original information to be fingerprinted. B's inputs are $x_R, \mathsf{sig}, s_1, s_2$, and $\mathsf{Cert}(Y \| x_R)$. The computations are performed as follows:
   i. $\mathsf{val}_1 = \mathsf{Verify}_1(\mathsf{text}, \mathsf{sig}, Y)$. The B-DH signature sig on text is verified by the anonymous public-key Y. The output val_1 is a Boolean variable only seen by M which is true if and only if the signature verification is completed successfully.

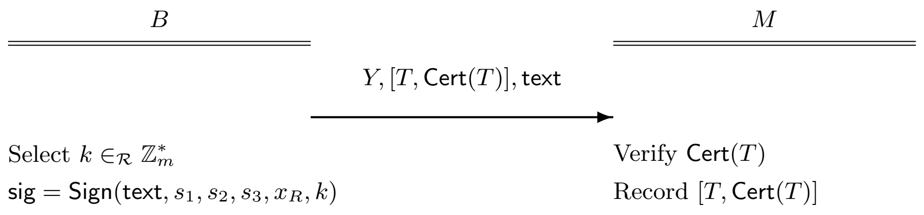

B
M
$Y, [T, \mathsf{Cert}(T)], \mathsf{text}$
Select $k \in_{\mathcal{R}} \mathbb{Z}_m^*$
$\mathsf{sig} = \mathsf{Sign}(\mathsf{text}, s_1, s_2, s_3, x_R, k)$
Verify $\mathsf{Cert}(T)$
Record $[T, \mathsf{Cert}(T)]$

Fig. 2. Buyer authentication of fingerprinting protocol FAuth_{MB}

ii. $\mathsf{val}_2 = \mathsf{Verify}_2(Y, \mathsf{Cert}(Y \| x_R), s_1, s_2, x_R, T)$. Firstly, the certificate $\mathsf{Cert}(Y \| x_R)$ on Y is verified. Secondly, it is checked whether $T = \hat{e}(s_1 s_2 P, x_R P)$. The output is also a Boolean variable only seen by M which is true if and only if the two aforementioned checks are completed successfully.

iii. $\mathsf{item}^\star = \mathsf{Fing}(\mathsf{item}, \mathsf{emb})$. A collusion-tolerant fingerprinting algorithm as used in [3,17] is applied to embed emb into the original information item, where

$$\mathsf{emb} = \mathsf{text} \| \mathsf{sig} \| Y \| \mathsf{Cert}(Y \| x_R) \| s_1 \| s_2 \| x_R \| T. \quad (1)$$

As a consequence, the fingerprinted information $\mathsf{item}^\star$ is obtained as output and is only seen by B. In the above two-party computation, M allows him to obtain outputs first and, unless val_1 and val_2 are both true, B does not get his output $\mathsf{item}^\star$.

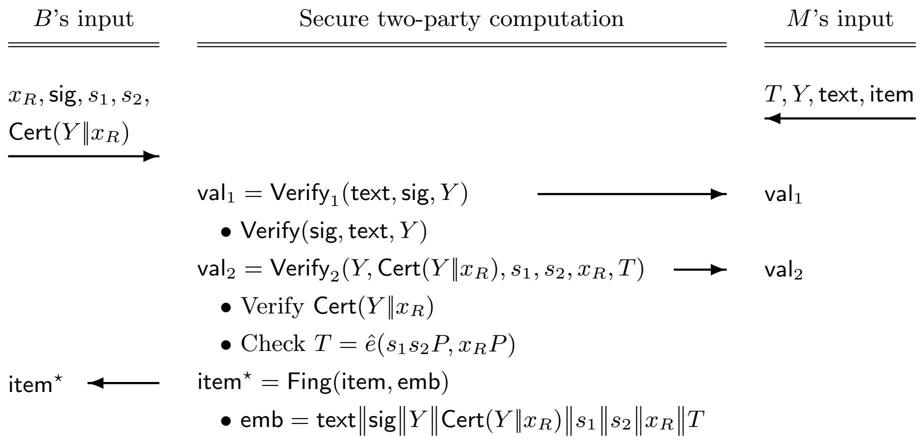
From the overall point of view, the fingerprinting protocol can be divided into two steps as following: buyer authentication and secure two-party computation. The former works as depicted on Figure 2, and the latter works as shown in Figure 3.

4.3 Identification Procedure

When M detects illegal redistribution of $\mathsf{item}^\star$, he performs the identification protocol on the ground of information extracted from $\mathsf{item}^\star$ and the purchase record. On finding an illegal copy redistributed, M extracts emb. The extracted information contains the values specified by Eq. (1) and is combined with the purchase record $[T, \mathsf{Cert}(T)]$ by M in order to provide a redistribution proof.

Protocol [identification] – FIden_{MR}.

1. The signature sig on text is verified using the pseudonym public-key Y.
2. The value x_R links the certificates T and Y. In addition, the value x_R cannot be altered since it is part of the certificates.
3. The value x_R proves that the owner of the pseudonym public-key Y is the same as the owner of T. This is because, according to the registration protocol, R only reveals x_R to B after B has provided such that $T = \hat{e}(s_1 s_2 P, x_R P)$. Therefore, provided that the $\mathsf{B\text{-}DH}$ problem is hard, B cannot produce a correct value T without knowing x_R in polynomial time.



B's input
Secure two-party computation
M's input
$x_R, \mathsf{sig}, s_1, s_2, \mathsf{Cert}(Y \| x_R)$
$T, Y, \mathsf{text}, \mathsf{item}$
$\mathsf{val}_1 = \mathsf{Verify}_1(\mathsf{text}, \mathsf{sig}, Y)$
• $\mathsf{Verify}(\mathsf{sig}, \mathsf{text}, Y)$
val_1
$\mathsf{val}_2 = \mathsf{Verify}_2(Y, \mathsf{Cert}(Y \| x_R), s_1, s_2, x_R, T)$
val_2
• Verify $\mathsf{Cert}(Y \| x_R)$
• Check $T = \hat{e}(s_1 s_2 P, x_R P)$
$\mathsf{item}^\star$
$\mathsf{item}^\star = \mathsf{Fing}(\mathsf{item}, \mathsf{emb})$
• $\mathsf{emb} = \mathsf{text} \| \mathsf{sig} \| Y \| \mathsf{Cert}(Y \| x_R) \| s_1 \| s_2 \| x_R \| T$

Fig. 3. Secure two-party computation of fingerprinting protocol FFing_{MB}

4. In consequence, in order to identify an illegally redistributing buyer, M attempts to raise the public keys of buyers to x_R such that $\hat{e}(Y, P) = y_B \cdot \hat{e}(P, P)^{x_R}$. Now the dishonest buyer has been identified. Note that x_R cannot be forged by M to unjustly accuse a buyer because T and Y are publicly certified.

5 Analysis of Security

We analyze in this section the security of the construction proposed in Section 4.

Theorem 1. *Under the B-DHIA, the protocol $\mathcal{P}^{\mathsf{B\text{-}DH}}_{\mathsf{fing}}$ is a secure* AAF *protocol.*

Proof.

The first condition of Definition 6 follows immediately from the description of $\mathcal{P}^{\mathsf{B\text{-}DH}}_{\mathsf{fing}}$. The theorem now follows from the following two lemmas.

Lemma 1. *Under the B-DHIA, let the* $(\mathsf{FKG}_R, \mathsf{FReg}_{RB})$ *be a sub-protocol of* $\mathcal{P}^{\mathsf{B\text{-}DH}}_{\mathsf{fing}}$*. Let Δ be a breaker against the* AAF *security of* $\mathcal{P}^{\mathsf{B\text{-}DH}}_{\mathsf{fing}}$ *within a time bound t and with at least success probability ε. Then there exists an adversary $\mathcal{A}$ that (τ, ϵ)-breaks the* B-DH *problem whose running time $\tau = O(t_{on} \cdot \varepsilon^{-1} + t_{off})$, and success probability*

$$\mathsf{Succ}^{\mathsf{B\text{-}DH}}_{\mathcal{IG}}(\mathcal{A}) \geq \frac{1}{16 \cdot \varepsilon},$$

where denote by t_{on} its on-line running time and by t_{off} its off-line running time.

Proof(sketch).

R only sees X, Y, and zero-knowledge proofs. It is clear that the zero-knowledge proofs leak no information on B. If we don't consider the zero-knowledge proofs, R requires knowledge of x_B to find the value Y' such that $Y' + T_R = y_B$. With considering the zero-knowledge proofs, then the breaker Δ without knowing x_B can compute X, Y such that $X = s_1 s_2 P$, $Y = x_B P + T_R$. Hence, the breaker can solve the B-DH problem, which contracts the B-DHIA. That is, the breaker Δ is reduced to the adversary $\mathcal{A}$. The detail of proof refers to [12].□

The registration protocol $(\mathsf{FKG}_R, \mathsf{FReg}_{RB})$ provides buyer authentication without compromising the private key x_B of B. This means that the protocol $\mathcal{P}_{\mathsf{fing}}^{\mathsf{B\text{-}DH}}$ meets the second condition of Definition 6.

Lemma 2. *Assume that a secure two-party computation on the* B-DH *problem is feasible. Under the B-DHIA, let the* $(\mathsf{FAuth}_{MB}, \mathsf{FFing}_{MB})$ *be a sub-protocol of* $\mathcal{P}_{\mathsf{fing}}^{\mathsf{B\text{-}DH}}$. *Let* Δ *be a breaker against the* AAF *security of* $\mathcal{P}_{\mathsf{fing}}^{\mathsf{B\text{-}DH}}$ *within a time bound* t *and with at least success probability* ε. *Then there exists an adversary* $\mathcal{A}$ *that* (τ, ϵ)*-breaks the* B-DH *problem whose running time* $\tau{=}O(t_{on} \cdot \varepsilon^{-1} + t_{off})$, *and success probability*

$$\mathsf{Succ}_{\mathcal{IG}}^{\mathsf{B\text{-}DH}}(\mathcal{A}) \geq \frac{1}{\tau \cdot \varepsilon},$$

where denote by t_{on} *its on-line running time and by* t_{off} *its off-line running time.*

Proof(sketch).
In the fingerprinting protocol, M knows Y, $[T, \mathsf{Cert}(T)]$, and outputs of a secure two-party computation that are val_1 and val_2. However, if the secure two-party computation is infeasible, the only way for M to know x_R is to solve the B-DH problem such that $T = \hat{e}(X, x_R P) = \hat{e}(P, P)^{s_1 s_2 x_R}$ using $\mathsf{Cert}(T)$. If it is possible to solve in polynomial-time bound, then the attacker $\mathcal{A}$ violates the B-DHIA. Therefore, the breaker Δ is reduced to the adversary $\mathcal{A}$. The detail of proof refers to [12]. □

An honest buyer who follows the fingerprinting protocol $(\mathsf{FAuth}_{MB}, \mathsf{FFing}_{MB})$ will not be identified if the secure two-party computation on the B-DH problem is feasible. This means that the protocol $\mathcal{P}_{\mathsf{fing}}^{\mathsf{B\text{-}DH}}$ meets the third condition of Definition 6.

In the sequel, we have the breaker Δ against the AAF security with respect to the adversary $\mathcal{A}$ with running time τ and at least success probability ϵ such that running time is bounded t, and the success probability

$$\mathsf{Succ}_{\mathcal{P}}^{\mathsf{AAF}}(\Delta) < \frac{1}{16 \cdot \varepsilon} + \frac{1}{\tau \cdot \varepsilon}.$$

This completes the proof of the theorem. □

6 Comparison

The proposal gains an advantage over previous schemes with respect to the key size because the proposal works on an elliptic curve. Therefore, our construction increases efficiency and, at the same time, decreases computation quantity.

Table 1. Computation Complexity

Protocol	Scheme [10]	Scheme [8]	Our scheme
Registration	$6\mathsf{E}$	$7\mathsf{E}$	$1\mathsf{S} + 1\mathsf{P}$
	$1\mathsf{M}$	$2\mathsf{M}$	$1\mathsf{A}$
Fingerprinting	$5\mathsf{E}$	$4\mathsf{E}$	$2\mathsf{P}$
	$1\mathsf{M}$	$2\mathsf{M}$	0
Identification	$3\mathsf{E} + N/2$	$(3+1)\mathsf{E}$	$1\mathsf{S} + 2\mathsf{P}$
	$2\mathsf{M}$	$3\mathsf{M}$	0

Table 2. Communication Complexity

Protocol	Scheme [10]	Scheme [8]	Our scheme
Registration	$4\mathsf{R}$	$2\mathsf{R}$	$3\mathsf{R}$
Fingerprinting	$6\mathsf{R}$	$6\mathsf{R}$	$6\mathsf{R}$
Identification	$\frac{N}{2}\mathsf{R}$	$\frac{N}{2}\mathsf{R}$	$\frac{N}{2}\mathsf{R}$

Table 1 and 2 show the comparison. We denote by E the cost of modular exponentiation, by M the cost of modular multiplication, by S the cost of the point multiplication on an elliptic curve, by P the cost the pairing on an elliptic curve, by A the cost of point addition on an elliptic curve, by R the number of rounds in given protocol, and by N the number of public key in directory.

7 Concluding Remarks

We proposed a practical protocol suitable for anonymous fingerprinting which is computationally much simpler than previous protocols.

As future works, firstly we have to replace the zero-knowledge proof by a more efficient protocol in the registration protocol. Secondly, we should realize the secure two-party computation on the $\mathsf{B\text{-}DH}$ problem used in the fingerprinting protocol.

References

1. D. Boneh and M. Franklin, "ID-based encryption from the Weil-pairing", *Advances in Cryptology – Crypto '2001*, LNCS 2139, Springer-Verlag, pp. 213–229, 2001.
2. G. R. Blakley, C. Meadows, and G. B. Purdy, "Fingerprinting long forgiving messages", *Advances in Cryptology – Crypto 1985*, LNCS 218, Springer-Verlag, pp. 180–189, 1986.
3. D. Boneh and J. Shaw, "Collusion-secure fingerprinting for digital data", *Advances in Cryptology – Crypto 1995*, LNCS 963, Springer-Verlag, pp. 452–465, 1995.
4. D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil-pairing", *Advances in Cryptology – Asiacrypt '2001*, LNCS 2248, Springer-Verlag, pp. 514–532, 2001.
5. J. Camenisch, "Efficient anonymous fingerprinting with group signatures", *Advances in Cryptology – Asiacrypt 2000*, LNCS 1976, Springer-Verlag, pp. 415–428, 2000.

6. J. Cha and J. Cheon, "An identity-based signature from gap Diffie-Hellman groups", Available from *http://eprint.iacr.org*, 2002.
7. D. Chaum, I. Damgård, and J. van de Graaf, "Multiparty computations ensuring privacy of each party's input and correctness of the result", *Advances in Cryptology – Crypto 1987*, LNCS 293, Springer-Verlag, pp. 87–119. 1988.
8. C. Chung, S. Choi, Y. Choi, and D. Won, "Efficient anonymous fingerprinting of electronic information with improved automatic identification of redistributors", *ICISC 2000*, LNCS 2015, Springer-Verlag, pp. 221-234, 2001.
9. J. Domingo-Ferrer, "Anonymous fingerprinting based on committed oblvious transfer", *PKC 1999*, LNCS 1560, Springer-Verlag, pp. 43–52, 1999.
10. J. Domingo-Ferrer, "Anonymous fingerprinting of electronic information with automatic identification redistributors", *IEE Electronics Letters*, Vol. 43, No. 13, 1998.
11. J. Domingo-Ferrer and H. Herrera-Joancomartí, "Efficient smart-card based anonymous fingerprinting", *Smart Card Research and Advanced Application – CARDIS 1998*, 1998.
12. M.S. Kim and K.J. Kim, "A new identification scheme base on the bilinear Diffie-Hellman problem", *ACISP 2002*, LNCS 2384, Springer-Verlag, pp. 362–378, 2002.
13. M. Kuribayashi and H. Tanaka, "A new anonymous fingerprinting scheme with high enciphering rate", *Indocrypt 2001*, LNCS 2247, Springer-Verlag, pp. 30–39, 2001.
14. B. Pfitzmann and M. Schunter, "Asymmetric fingerprinting", *Advances in Cryptology – Eurocrypt 1996*, LNCS 1070, Springer-Verlag, pp. 84–95, 1996.
15. B. Pfitzmann and M. Waidner, "Anonymous fingerprinting", *Advances in Cryptology – Eurocrypt 1997*, LNCS 1233, Springer-Verlag, pp. 88–102, 1997.
16. A.-R. Sadeghi, "How to break a semi-anonymous fingerprinting scheme", *Information Hiding, 4th International Workshop 2001*, LNCS 2137, Springer-Verlag, pp. 384–394, 2001.
17. W. Trappe, M. Wu, and K.J. R. Liu, "Collusion-resistant fingerprinting for multimedia",IEEE International Conference on *Acoustics, Speech, and Signal Processing*, Vol. 4, pp. 3309–3312, 2002.
18. H. Yoshiura, R. Sasaki, and K. Takaragi, "Secure fingerprinting using public-key cryptography", *Security Protocols-6th International Workshop*, LNCS 1550, Springer-Verlag, pp. 83–89, 1998.

Reducing the Memory Complexity of Type-Inference Algorithms

David Naccache[1], Alexei Tchoulkine[1], Christophe Tymen[1], and Elena Trichina[2]

[1] Gemplus Card International
34 rue Guynemer, Issy-les-Moulineaux, F-92447, France
{david.naccache,alexei.tchoulkine,christophe.tymen}@gemplus.com
[2] University of Kuopio
Department of Computer Science and Applied Mathematics
Po.B. 1627, FIN-70211, Kuopio, Finland
elena.trichina@cs.uku.fi

Abstract. In the Java Virtual Machine, the byte-code verifier checks low-level security properties that ensure that the downloaded code cannot bypass the virtual machine's security mechanisms. One of the statically ensured properties is *type safety.* The type-inference phase is the overwhelming resource-consuming part of the verification process.
This paper addresses the RAM bottleneck met while verifying mobile code in memory-constrained environments such as smart-cards. We propose to modify the algorithm in a way that significantly reduces memory consumption.

1 Introduction

The Java Card architecture for smart cards allows new applications, called *applets*, to be downloaded into smart-cards. While bringing considerable flexibility and extending the horizons of smart-card usage this *post issuance* feature raises major security issues. Upon their loading, malicious applets can try to subvert the JVM's security in a variety of ways. For example, they might try to overflow the stack, hoping to modify memory locations which they are not allowed to access, cast objects inappropriately to corrupt arbitrary memory areas or even modify other programs (Trojan horse attacks). While the general security issues raised by applet download are well known, transferring Java's safety model into resource-constrained devices such as smart-cards appears to require the devising of delicate security-performance trade-offs.

Upon download, an applet's byte-code is subject to a static analysis called *byte-code verification* which purpose is to make sure that the applet's code is well-typed. This is necessary to ascertain that the code will not attempt to violate Java's security policy by performing ill-typed operations at runtime (*e.g.* forging object references from integers or calling directly API private methods). Today's *de facto* verification standard is Sun's algorithm [6] which has the advantage of being able to verify any class file resulting from any standard compilation chain. While the time and space complexities of Sun's algorithm suit personal computers, the memory complexity of this algorithm appears prohibitive for smart-cards, where RAM is a significant cost-factor.

This limitation gave birth to a number of innovating workarounds [4,5,8]. The work reported in this paper describes a new memory-optimization technique.

The rest of the paper is organized as follows: the next section recalls Java's security model and Sun's verification algorithm with a specific focus on its *data-flow analysis* part. The subsequent sections describe in detail our algorithm which benchmarks are given in the last section.

2 Java Security

The *Java Virtual Machine (JVM) Specification* [6] defines the executable file structure, called the *class file* format, to which all Java programs are compiled. In a class file, the executable code of *methods* (Java methods are the equivalent of C functions) is found in *code-array* structures. The executable code and some method-specific runtime information (namely, the maximal operand stack size $S_{\max}$ and the number of local variables $L_{\max}$ claimed by the method) constitute a *code-attribute*. We briefly overview the general stages that a Java code goes through upon download.

To begin with, the classes of a Java program are translated into independent class files at compile-time. Upon a load request, a class file is transferred over the network to its recipient where, at link-time, symbolic references are resolved. Finally, upon method invocation, the relevant method code is interpreted (run) by the JVM.

2.1 Byte-Code Verification

Byte-code verification [3] is a link-time phase where the method's run-time behavior is proved to be *semantically correct*. The *byte-code* is the executable sequence of bytes of the code-array of a method's code-attribute.

As this ends normally, the receiver assumes that the analyzed file complies with the general syntactical description of the class file format.

Then, a second verification step ascertains that the code will only manipulate values which types are compatible with Java's safety rules. This is achieved by a type-based *data-flow analysis* which abstractly executes the method's byte-code, by modeling the effect of the successive byte-codes on the *types* of the variables read or written by the code.

The next section explains the semantics of *type checking*, *i.e.*, the process of verifying that a given pre-constructed type is correct with respect to a given class file. We explain why and how such a type can always be constructed and describe the basic idea behind data-flow analysis.

The Semantics of Type Checking. A natural way to analyze the behavior of a program is to study its effect on the machine's memory. At runtime, each program point can be looked upon as a memory *instruction frame* describing the set of all the runtime values possibly taken by the JVM's stack and local variables.

Since run-time information, such as actual input data is unknown before execution starts, the best an analysis may do is reason about *sets* of possible computations. An essential notion used for doing so is the *collecting semantics* defined in [2] where, instead of computing on a full *semantic* domain (values), one computes on a restricted *abstract* domain (types).

For reasoning with types, one must precisely classify the information expressed by types. A natural way to determine how (in)comparable types are is to rank all types in a *lattice* $\mathcal{L}$.

The most general type is called *top* and denoted $\top$. $\top$ represents the *potential simultaneous presence of all types*, *i.e.* the *absence of (specific) information*. By definition, a special null-pointer type (denoted `null`) terminates the inheritance chain of all object descendants. Formally, this defines a pointed complete partial order (CPO) $\preceq$ on the lattice $\mathcal{L}$.

Stack elements and local variable types are hence tuples of elements of $\mathcal{L}$ to which one can apply *point-wise ordering*.

$$\mathcal{L} = \begin{tikzcd}
 & & \top \arrow[dll] \arrow[d] \arrow[drr] & & & & & \\
\texttt{int} & & \cdots & & \texttt{Object} \arrow[dll] \arrow[d] \arrow[drr] & & & \\
 & & \tau_1 \arrow[dl] \arrow[d] \arrow[dr] & & \cdots & & \tau_k \arrow[dl] \arrow[d] \arrow[dr] & \\
 & \vdots & \cdots \arrow[dd] & \vdots & & \vdots & \cdots \arrow[dd] & \vdots \\
 & \tau_{\ldots} \arrow[d] & & \tau_{\ldots} \arrow[d] & & \tau_{\ldots} \arrow[d] & & \tau_{\ldots} \arrow[d] \\
 & \texttt{null} & \texttt{null} & \texttt{null} & & \texttt{null} & \texttt{null} & \texttt{null}
\end{tikzcd}$$

Abstract Interpretation. The verification process described in [6] 4.9, is an (iterative data-flow analysis) algorithm that attempts to builds an *abstract description* of the JVM's memory for each program point. A byte-code is safe if the construction of such an abstract description succeeds.

Assume, for example, that an `iadd` is present at some program point. The `i` in `iadd` hints that this instruction operates on integers. `iadd`'s effect on the JVM is indeed very simple: the two topmost stack elements are popped, added and the sum is pushed back into the stack. An abstract interpreter will disregard the arithmetic meaning of `iadd` and reason with types: `iadd` pops two `int` elements from the stack and pushes back an `int`. From an abstract perspective, `iadd` and `isub` have identical effects on the JVM.

As an immediate corollary, a valid stack for executing an `iadd` *must* have a value which can be abstracted as `int.int.`S, where S may contain any sequence of types (which are irrelevant for the interpretation of our `iadd`). After executing `iadd` the stack becomes `int.`S

Denoting by L the JVM's local variable area (irrelevant to `iadd`), the total effect of `iadd`'s abstract interpretation on the JVM's memory can be described by the *transition rule* ϑ:

$$\texttt{iadd}: \quad (\texttt{int.int.}S, \quad L) \mapsto (\texttt{int.}S, \quad L)$$

The following table defines the transition rules of seven representative JVM instructions[1].

[1] Note that the test $n \in L$ is equivalent to ascertaining that $0 \leq n \leq L_{\max}$.

Instruction	Transition rule ϑ	Security test
iconst[n]	$(S, \quad L) \mapsto (\texttt{int}.S, \quad L)$	$\mid S \mid < S_{\max}$
iload[n]	$(S, \quad L) \mapsto (\texttt{int}.S, \quad L)$	$n \in L$, $L[n] == \texttt{int}$, $\mid S \mid < S_{\max}$
istore[n]	$(\texttt{int}.S, \quad L) \mapsto (S, \quad L\{n \to \texttt{int}\})$	$n \in L$
aload[n]	$(S, \quad L) \mapsto (L[n].S, \quad L)$	$n \in L$, $L[n] \preceq \texttt{Object}$, $\mid S \mid < S_{\max}$
astore[n]	$(\tau.S, \quad L) \mapsto (S, \quad L\{n \to \tau\})$	$n \in L$, $\tau \preceq \texttt{Object}$
dup	$(\tau.S, \quad L) \mapsto (\tau.\tau.S, \quad L)$	$\mid S \mid < S_{\max}$
getfield $C.f.\tau$	$(ref(D).S, \quad L) \mapsto (\tau.S, \quad L)$	$D \preceq C$

For the first instruction of the method, the local variables that represent parameters are initialized with the types τ_j indicated by the method's signature; the stack is empty (ϵ) and all other local variables are filled with $\top$s. Hence, the initial frame is set to:

$$(\epsilon, \quad (\texttt{this}, \tau_1, \ldots, \tau_{n-1}, \top, \ldots, \top))$$

For other instructions, no information regarding the stack or the local variables is available.

Verifying a method whose body is a straight-line code (no branches), is easy: we simply iterate the abstract interpreter' transition function ϑ over the successive instructions, taking the stack and register types *after* any given instruction as the stack and register types *before* the next instruction. The types describing the successive JVM memory-states produced by the successive instructions are called *frames*.

Denoting by in(i) the frame *before* instruction i and by out(i) the frame *after* instruction i, we get the following data-flow equation where evaluation starts from the right:

$$\text{in}(i+1) \leftarrow \text{out}(i) \leftarrow \vartheta_i(\text{in}(i))$$

Branches introduce forks and joins into the method's flowchart. By extension, if an instruction i has several predecessors with different exit frames, i's frame is computed as the *least common ancestor*[2] (LCA) of all the predecessors' exit frames:

$$\text{in}(i) = \text{LCA}\{\text{out}(i) \mid j \in \text{Predecessor}(i)\}.$$

Finding an assignment of frames to program points which is sufficiently conservative for all execution paths requires testing them all; this is what the verification algorithm does. Whenever some in(i) is adjusted, all frames in(j) that depend on in(i) have to be adjusted too, causing additional iterations until a *fix-point* (*i.e.*, no more adjustments are required) is reached. The final set of frames is a *proof* that the verification terminated with success. In other words, that the byte-code is *well-typed*. We refer the reader to the verification algorithm described in [6] page 143 (section 4.9.2) which summarizes the verification process. This algorithm is denoted hereafter $\mathcal{V}_{\text{sun}}$.

2.2 Basic Blocks

As explained above, the data-flow type analysis of a straight-line code consists of simply applying the transition function to the sequence of instructions $i_1, i_2, ..., i_t$

[2] The LCA operation is frequently called *unification*.

taking $\text{in}(i_k) \leftarrow \text{out}(i_{k-1})$. This property can be used for optimizing the algorithm.

Following [1,7], we call a *basic block* ($\mathbb{B}$) a straight-line sequence of instructions that can be entered only at its beginning and exited only at its end.

In several implementations of Sun's algorithm, the data-flow equations evolve at the basic-block-level rather than at the instruction-level. In other words, it suffices to keep track in permanent memory only the frames $\text{in}(\ell)$ where ℓ is the first instruction of a $\mathbb{B}$ (*i.e.*, a branch target). All other frames within a basic block can be temporarily recomputed on the fly. By extension, we denote by $\text{in}(\mathbb{B})$ and $\text{out}(\mathbb{B})$, the frames before and after the execution of $\mathbb{B}$. The entire program is denoted by $\mathbb{P}$.

3 A Memory-Constrained Version of Sun's Algorithm

Denoting by N_{blocks} the number of $\mathbb{B}$s in a method, a straightforward implementation of Sun's algorithm allocates N_{blocks} frames, each of size $L_{\max} + S_{\max}$.

$L_{\max}$ and $S_{\max}$ are determined by the compiler and appear in the method's header. This results in an $\mathcal{O}((L_{\max} + S_{\max}) \times N_{\text{blocks}})$ memory-complexity. In practice, the verification of moderately complex methods would frequently require a few thousands of bytes.

A property of Java code is that a *unique* stack height is associated to each program point. This property is actually verified on the fly during type-inference, although it could be perfectly checked *independently* of type-inference.

In other words, the computation of stack heights throughout execution does not require the modeling of the instructions' effect on types, but only on the stack-pointer.

Denoting by σ_i the stack height at the beginning of $\mathbb{B}_i$, one can allocate for the stack only σ_i RAM cells in $\text{in}(\mathbb{B}_i)$, knowing for sure that the verifier will never attempt to enter $\mathbb{B}_i$ with more (or less) than σ_i stack levels.

However, during $\mathbb{B}_i$'s abstract interpretation, the stack may grow higher than σ_i. To cope with this, one working buffer of $S_{\max}$ RAM cells is enough. Hence, the total amount of RAM required for stack manipulations is:

$$S_{\max} + \sum_{i=0}^{N_{\text{blocks}}} \sigma_i$$

Considering that the stack is normally empty at most jump targets [4], this trivial optimization turns out to be significant. Taking a 'moderately complex' example from [4] where $S_{\max} = 5$, $N_{\text{blocks}} = 50$ and $L_{\max} = 15$, we get a 30% memory saving.

Can this idea be generalized to local variables? In other words, can one exploit the fact all $\mathbb{B}$s do not necessarily use all the local variables? In the `Toy` example below[3] the compiler used three registers, namely, $L[1]$ (parameter n, declared

[3] For the sake of simplicity we ignore the `this` argument.

as `int`), $L[2]$ (variable m) and $L[3]$ (variable q). Only $\mathbb{B}_1$, uses all three local variables ($L[1]$, $L[2]$, $L[3]$). The three other blocks use only one variable each: $\mathbb{B}_0$ and $\mathbb{B}_3$ use only $r[2]$ and $\mathbb{B}_2$ uses only $L[1]$. In the example $S_{\max} = 2$.

As mentioned above, a straightforward implementation of Sun's algorithm would allocate for `Toy` four RAM contexts, each of size $S_{\max} + L_{\max} = 2 + 3$, *i.e.* altogether $5 \times 4 = 20$ RAM registers. Should we manage to modify Sun's algorithm so that the frame of each $\mathbb{B}$ would contain only as many registers as local variables used in this block and σ_i stack cells, the total memory usage would melt down by 60% to $2 + 1 + 3 + 1 + 1 = 8$ RAM cells. Moreover, can we hope to keep in $\mathbb{B}$'s frame only the stack chunk used effectively in $\mathbb{B}$? The answer is affirmative, as we will see in sections 4 and 5.

```
Class Toy extends Object {
   int toy (int n) {
       int m; int q;
       m = 0;
       while (n>0) {
           n = n-1;
           q = 1;
           m = m + q;
       }
   return m;
   }
}
```

$\xrightarrow{\text{compile}}$

code	$L[1]$	$L[2]$	$L[3]$
$\mathbb{B}_0$		√	
$\mathbb{B}_0$ `0 iconst_0`			
$\mathbb{B}_0$ `1 istore_2`			
$\mathbb{B}_0$ `3 goto 22`			
$\mathbb{B}_1$	√	√	√
$\mathbb{B}_1$ `6 iload_1`			
$\mathbb{B}_1$ `8 iconst_1`			
$\mathbb{B}_1$ `9 isub`			
$\mathbb{B}_1$ `10 istore_1`			
$\mathbb{B}_1$ `12 iconst_1`			
$\mathbb{B}_1$ `13 istore_3`			
$\mathbb{B}_1$ `15 iload_2`			
$\mathbb{B}_1$ `17 iload_3`			
$\mathbb{B}_1$ `19 iadd`			
$\mathbb{B}_1$ `20 istore_2`			
$\mathbb{B}_2$	√		
$\mathbb{B}_2$ `22 iload_1`			
$\mathbb{B}_2$ `24 ifgt 6`			
$\mathbb{B}_3$		√	
$\mathbb{B}_3$ `27 iload_2`			
$\mathbb{B}_3$ `29 ireturn`			

4 Exploring the Stack's Behavior

As we saw, a unique stack height is associated to each program point; consequently, a particular stack height σ_i is associated to the entry point of each $\mathbb{B}_i$. During the interpretation of $\mathbb{B}_i$, elements are pushed and popped, causing the stack to vary between two $\mathbb{B}_i$-specific bounds $\underline{s}_i$ and $\overline{s}_i$. Note that $0 \leq \underline{s}_i \leq \sigma_i \leq \overline{s}_i \leq S_{\max} = \max\{\overline{s}_i\}$.

This section presents a simple algorithm for computing $\{\underline{s}_0, \sigma_0\}, \{\underline{s}_1, \sigma_1\}, \ldots$ from $\mathbb{B}_0, \mathbb{B}_1, \ldots$

The algorithm uses a table Δ associating to each instruction a signed integer indicating the effect of this instruction on the stack's size:

Δ	**Instruction**	Δ	**Instruction**	Δ	**Instruction**	Δ	**Instruction**
2	`iconst`[n]	1	`sconst`[n]	1	`bspush`	2	`bipush`
1	`aload`	1	`sload`	1	`aload`[n]	2	`iload`[n]
-1	`aaload`	0	`iaload`	-1	`astore`	-2	`istore`
-1	`astrore`[n]	-2	`store`[n]	-1	`pop`	1	`dup`
-1	`sadd,smul`	-2	`iadd,imul`	0	`getfield_a`	1	`getfield_i`
0	`iinc`	-3	`icmp`	-1	`ifne`	-2	`if_acmpne`
0	`goto`	0	`return`	0	`athrow`	0	`arraylength`

The information we are looking for is easily obtained by running Sun's algorithm with the modeling effect on types *turned off*, monitoring only the code's effect on the stack pointer.

Algorithm PredictStack($\mathbb{P}$).

```
– Associate to each $\mathbb{B}_i$ a bit changed[$i$] indicating if this $\mathbb{B}$ needs to be re-examined;
  initialize all the changed[$i$]-bits to zero.
– Set $\sigma_0 \leftarrow 0$; changed[0] $\leftarrow 1$;
– While $\exists\ \ \mathbb{B}_i$ such that changed[$i$] $== 1$:
    • Set $\underline{s}_i \leftarrow \sigma_i$; $\alpha \leftarrow \sigma_i$; changed[$i$] $\leftarrow 0$;
    • Let $j_1, j_2, ..., j_t$ be the successive instructions of $\mathbb{B}_i$.
        ∗ For $m \leftarrow 1$ to $t$
            · $\alpha \leftarrow \alpha + \Delta(j_m)$
            · If $0 \leq \alpha \leq S_{\max}$ then $\underline{s}_i \leftarrow \min(\underline{s}_i, \alpha)$ else report a failure.
        ∗ If $i == N_{\text{blocks}} - 1$ and it is possible to 'fall off' instruction $j_t$ then report
          a failure.
    • For each successor block $\mathbb{B}_k$ of $\mathbb{B}_i$ :
        ∗ If $\mathbb{B}_k$ is visited for the first time, set $\sigma_k \leftarrow \alpha$; changed[$k$] $\leftarrow 1$
        ∗ If $\mathbb{B}_k$ was visited before and $\sigma_k \neq \alpha$, then report a failure.
– Return $\{\underline{s}_0, \sigma_0\}, \{\underline{s}_1, \sigma_1\}, \ldots$
```

5 Memory-Constrained Local Variable Verification

Definition. The *used-frame* associated to $\mathbb{B}_i$ is a memory space u_in($\mathbb{B}_i$) representing the stack chunk $[\underline{s}_i, \ldots, \sigma_i]$ and the local variables actually used (read or written to) during the execution of $\mathbb{B}_i$.

What would happen if one would run Sun's algorithm while unifying only the memory elements present in the used-frame of each $\mathbb{B}_i$?

Unfortunately, safety is not preserved, as is obvious from the following example where the type information assigned to r_2 by $\mathbb{B}_0$ will never reach $\mathbb{B}_3$ (r_2 is nonetheless essential for the abstract interpretation of $\mathbb{B}_3$):

$$\begin{tikzcd}
 & \text{start} \arrow[d] & \\
 & \begin{array}{c}\mathbb{B}_0\\ \text{uses } r_1, r_2\end{array} \arrow[dl] \arrow[dr] & \\
\begin{array}{c}\mathbb{B}_1\\ \text{uses only } r_1\end{array} \arrow[dr] & & \begin{array}{c}\mathbb{B}_2\\ \text{uses only } r_1\end{array} \arrow[dl] \\
 & \begin{array}{c}\mathbb{B}_3\\ \text{uses } r_1, r_2\end{array} \arrow[d] & \\
 & \text{exit} &
\end{tikzcd}$$

In the next section we remedy to this by adapting the algorithm as follows: *upon exiting a current block, we find all the 'first-hand' users for every variable belonging to the current used-frame, and perform the unification.*

By doing so, as a new block is reached (from any of its predecessors), all variables in its used-frame have been already unified, and we can simply start running a straight-line abstract interpretation. Let us formalize the solution.

5.1 v-Successor Blocks

We define a notion of the '*v-successor blocks*' of a block $\mathbb{B}_i$ and describe an algorithm for determining the v-successor blocks for a given i and v.

Definition. Let $\mathbb{B}_i$ be a basic block that uses variable v. $\mathbb{B}_j$ is a *v-successor block of* $\mathbb{B}_i$ if:

1. $\mathbb{B}_j$ uses (reads or writes) variable v.
2. There is a path from $\mathbb{B}_i$ to $\mathbb{B}_j$ in the method's control graph such that after $\mathbb{B}_i$ used v and before $\mathbb{B}_j$ used v, no other block on this path used v.

In essence, the v-successors of $\mathbb{B}_i$ are the *first consumers* of the value stored in v by $\mathbb{B}_i$. The v-successors of $\mathbb{B}_i$ can be computed by the following algorithm where $0 \le i < N_{\text{blocks}}$ and $v \in \text{u\_in}(\mathbb{B}_i)$:

Algorithm vSuccessors$(i, v, \mathbb{P})$.

```
– Initialize three $N_{\text{blocks}}$-bit arrays marked, visited and found, to zero.
– marked[$i$] $\leftarrow 1$
– While $\exists$ $k$ such that marked[$k$] $== 1$,
    • marked[$k$] $\leftarrow 0$
    • For all successors $\mathbb{B}_j$ of $\mathbb{B}_k$ for which visited[$j$] $== 0$
        * If $\mathbb{B}_j$ uses variable $v$ then found[$j$] $\leftarrow 1$ else marked[$j$] $\leftarrow 1$
        * visited[$j$] $\leftarrow 1$
– Return the bit array found.
```

found is such that found[j] $== 1$ iff $\mathbb{B}_j$ is a v-successor of $\mathbb{B}_i$.

5.2 Putting the Pieces Together

We now present our new verification algorithm, denoted $\mathcal{V}_{\text{new}}$. For convenience, we introduce a universal type, denoted by $\perp$, which represents the lowest possible node in $\mathcal{L}$. We denote by $\mathbb{B}_{-1}$ an empty block which does not contain any instructions. $\mathbb{B}_{-1}$'s used-frame contains by convention all the local variables and all the stack elements. $\mathbb{B}_{-1}$'s abstract interpretation is defined by $\text{out}(-1) \leftarrow \text{in}(-1)$. The successor of $\mathbb{B}_{-1}$ is $\mathbb{B}_0$.

$\mathcal{V}_{\text{new}}$ uses two $N_{\text{blocks}} + 1$-bit arrays changed_frame and changed_block , which indices run from -1 to $N_{\text{blocks}} - 1$.

The initialization phase of the algorithm consists of the following steps.

1. Initialize the used-frame u_in[-1] of $\mathbb{B}_{-1}$ by setting the local variables in u_in[-1] that correspond to the method's parameters to the types declared by the method's signature. Initialize all other local variables in u_in[-1] to $\top$. Initialize the stack elements in u_in[-1] to $\top$.
2. Run algorithm PredictStack$(\mathbb{P})$ to compute:

$$\{\underline{s}_0, \sigma_0\}, \{\underline{s}_1, \sigma_1\}, \ldots, \{\underline{s}_{N_{\text{blocks}}-1}, \sigma_{N_{\text{blocks}}-1}\}$$

3. For $i \leftarrow 0, \cdots, N_{\text{blocks-1}}$
   (a) build `u_in`$[i]$.
   (b) initialize all variables in `u_in`$[i]$ to $\perp$.
4. Set the arrays `changed_frame` and `changed_block` to zero.
5. Mark block $\mathbb{B}_{-1}$ by setting `changed_frame`$[-1]$ and `changed_block`$[-1]$ to one.

Next we execute the following loop, until array `changed_block` is entirely equal to zero.

1. Select an index i such that `changed_block`$[i] == 1$ and set `changed_block`$[i] \leftarrow 0$.
2. Model the effect of $\mathbb{B}_i$'s execution on the used-frame `u_in`$[i]$. Let `u_out`$[i]$ denote the resulting frame.
3. If the modeling exits without a failure, for every variable v in `u_out`$[i]$ do:
   (a) Determine the v-successors of $\mathbb{B}_i$ by running algorithm $\mathsf{vSuccessors}(i, v, \mathbb{P})$.
   (b) For each v-successor $\mathbb{B}_k$ of $\mathbb{B}_i$,
      i. unify variable v in `u_in`$[k]$ with variable v in `u_out`$[i]$,
      ii. if the type of v in `u_in`$[k]$ has changed,
         – set `changed_frame`$[k] \leftarrow 1$
         – for all blocks $\mathbb{B}_j$ with $j \neq i$ belonging to any path from block $\mathbb{B}_i$ to block $\mathbb{B}_k$, set `changed_frame`$[j] \leftarrow 1$.
4. For each successor $\mathbb{B}_j$ of $\mathbb{B}_i$, if `changed_frame`$[j] == 1$, set `changed_block`$[j] \leftarrow 1$. If this is the first time $\mathbb{B}_j$ is visited, set `changed_block`$[j] \leftarrow 1$.
5. Set `changed_frame`$[i] \leftarrow 0$.
6. Go to step 1.

5.3 Equivalence of $\mathcal{V}_{\text{new}}$ and $\mathcal{V}_{\text{sun}}$

In this section, we prove that a program is accepted by $\mathcal{V}_{\text{sun}}$ if and only if it is accepted by $\mathcal{V}_{\text{new}}$.

We first need to introduce some definitions. We denote by $\mathbb{P}$ a program, involving N basic blocks $\mathbb{B}_0, \ldots, \mathbb{B}_{N-1}$. Let $S = (i_1, \ldots, i_k, \ldots)$ be a sequence of integers. We say that S is an *admissible execution for* $(\mathbb{P}, \mathcal{V}_{sun})$ *(resp. for* $(\mathbb{P}, \mathcal{V}_{new})$*)* if

– there exists an integer n such that for $1 \leq j \leq n$, all the i_j are in $\{0, \ldots, N-1\}$.
– the successive verification of the blocks $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_n}$ is a possible order of execution of $\mathcal{V}_{\text{sun}}$ (resp. $\mathcal{V}_{\text{new}}$) when verifying $\mathbb{P}$.

Let M denote the maximal size of the stack during the execution of $\mathbb{P}$, added to the number of local variables used by $\mathbb{P}$. Given an admissible execution $(i_j)_{j \geq 1}$ of $(\mathbb{P}, \mathcal{V}_{\text{sun}})$, we define $F_{i_1,\ldots,i_k}(\mathcal{V}_{\text{sun}})$ the $M \times N$-tuple of types corresponding to the frames at the beginning of the blocks in $\mathcal{V}_{\text{sun}}$, resulting from the successive verification of the blocks $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_k}$. Similarly, we denote by $\overline{F}_{i_1,\ldots,i_k}(\mathcal{V}_{\text{new}})$ (resp. $\underline{F}_{i_1,\ldots,i_k}(\mathcal{V}_{\text{new}})$) the $M \times N$-tuple of types corresponding to the frames at the beginning of the blocks in $\mathcal{V}_{\text{new}}$, resulting from the successive verification of the blocks $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_k}$, where the variables missing are arbitrarily set to $\top$ (resp. to $\perp$).

We now define a modification of the algorithm $\mathcal{V}_{\rm sun}$, which we denote by $\mathcal{V}'_{\rm sun}$. $\mathcal{V}'_{\rm sun}$ is defined exactly as $\mathcal{V}_{\rm sun}$ in paragraph 4.9.2. of [6], except that step 4 is replaced by the following step 4':

4'. Unify out(i) with the in$(\cdot)$-frame of each successor $\mathbb{B}_j$.

- If $\mathbb{B}_j$ is visited for the first time,
  - record that out(i) calculated in steps 2 and 3 is now the in$(\cdot)$-frame of $\mathbb{B}_j$;
  - mark the successor instruction by setting the 'changed' bit.
- If $\mathbb{B}_j$ has been visited before,
  - Determine the set $\mathcal{V}_j$ of all the variables and all the stack elements v such that there exists a block $\mathbb{B}_k$ (possibly $\mathbb{B}_j$ itself) using v reachable from $\mathbb{B}_j$.
  - Unify out(i) with the successor instruction's in(j)-frame and update : in$(j) \leftarrow \mathrm{LCA}(\mathrm{in}(j), \mathrm{out}(i))$.
  - If the unification caused modifications in in(j) of at least one variable or stack element belonging to $\mathcal{V}_j$, mark $\mathbb{B}_j$ by setting its 'changed' bit.

Basically, $\mathcal{V}'_{\rm sun}$ marks as 'changed' only the successor blocks where the unification has affected the type of a variable which could be used later on in the execution flow. If the unification has affected only the type of variables that *cannot* be used later, the 'changed' bit is not set.

Under these notations, we have the following lemma:

Lemma 1. *If a program is accepted by $\mathcal{V}'_{sun}$, then it is accepted by $\mathcal{V}_{sun}$.*

Proof. Let $\mathbb{P}$ be a program accepted by $\mathcal{V}'_{\rm sun}$, and consider the the block-frames in(i) when $\mathcal{V}'_{\rm sun}$ stops, along with an admissible execution $S = (i_1, \ldots, i_k)$ leading to this state. As S is an admissible execution for $(\mathbb{P}, \mathcal{V}_{\rm sun})$ one can execute $\mathcal{V}_{\rm sun}$ starting from this point, setting all the 'changed' bits initially to one. From the definition of $\mathcal{V}'_{\rm sun}$, it is clear that a 'changed' bit is re-set to one during this execution of $\mathcal{V}_{\rm sun}$ iff a variable v in in(i) has been altered by a unification with out(j), and v is never used in any block reachable from $\mathbb{B}_i$. Consequently, the only modifications that occur in the block-frames concern unused variables, and thus cannot cause any verification failure. Thus, $\mathcal{V}_{\rm sun}$ reaches a fix-point. □

The role played by $\mathcal{V}'_{\rm sun}$ in the proof stems from the following lemma:

Lemma 2. *Let $S = (i_1, \ldots, i_k, \ldots)$ be an admissible execution of $(\mathbb{P}, \mathcal{V}'_{sun})$. Then*

1. *S is an admissible execution of $(\mathbb{P}, \mathcal{V}_{new})$.*
2. *For all $k \geq 1$,*
$$\overline{F}_{i_1,\ldots,i_k}(\mathcal{V}_{new}) \succeq F_{i_1,\ldots,i_k}(\mathcal{V}'_{sun}) \ .$$

Proof. The second part of the lemma is a trivial consequence of the first. For the first statement, we proceed by induction on the number of steps of the verification.

Initialization. Both for $\mathcal{V}_{\rm new}$ and $\mathcal{V}'_{\rm sun}$, the block to be verified after $\mathbb{B}_0$ can be any successor of $\mathbb{B}_0$.

Propagation. Let us assume that $(i_1, \ldots, i_n)$ is an admissible execution for $(\mathbb{P}, \mathcal{V}'_{\text{sun}})$, and let us consider that we have verified the blocks $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_n}$ following $\mathcal{V}'_{\text{sun}}$. We denote by $j_1, \ldots, j_l$ the indices of the blocks marked as 'changed' at that point, just before the execution of step 4 for block $\mathbb{B}_{i_n}$. We denote by $k_1, \ldots k_m$ the indices of the blocks that are newly marked as 'changed' after the execution of step 4 for the block $\mathbb{B}_{i_n}$. As $(i_1, \ldots, i_n)$ has length n, it is an admissible execution for $(\mathbb{P}, \mathcal{V}_{\text{new}})$. For the same reason, $(i_1, \ldots, i_{n-1}, j_a)$ is an admissible execution for $(\mathbb{P}, \mathcal{V}_{\text{new}})$, for all $1 \leq a \leq l$. Consequently, after having verified $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_n}$ by running $\mathcal{V}_{\text{new}}$, the bits $\texttt{changed\_block}[j_1], \ldots, \texttt{changed\_block}[j_l]$ are set to one. It remains to show that this is also the case for

$$\texttt{changed\_block}[k_1], \ldots, \texttt{changed\_block}[k_m]$$

We must distinguish two cases: if it is the first time $\mathbb{B}_{k_1}$ has been visited in the execution of $\mathcal{V}'_{\text{sun}}$, then it is also the case in $\mathcal{V}_{\text{new}}$, and thus, $\texttt{changed\_block}[k_1]$ will be set to one in $\mathcal{V}_{\text{new}}$. Otherwise, as $\mathbb{B}_{k_1}$ is newly marked as 'changed' in $\mathcal{V}'_{\text{sun}}$, there exists a variable (or a stack element) v such that

$$\text{u\_out}(i_n)_v \succ \text{u\_in}(k_1)_v$$

in $\mathcal{V}'_{\text{sun}}$. Let us assume now that all the $\mathbb{B}_{i_1}, \ldots, \mathbb{B}_{i_n}$ do not read nor write v. Then $\text{in}(k_1)_v$ and $\text{out}(i_n)_v$ should be equal, which is impossible. Consequently, there exists some $1 \leq b \leq n$ such that v is read or written by $\mathbb{B}_{i_b}$. Let us consider the greatest possible b. According to the definition of $\mathcal{V}'_{\text{sun}}$, $\mathbb{B}_{k_1}$ necessarily belongs to a path from $\mathbb{B}_{i_b}$ to a v-successors of $\mathbb{B}_{i_b}$. Thus, in $\mathcal{V}_{\text{new}}$, $\texttt{changed\_frame}[k_1] = 1$, which implies that $\texttt{changed\_block}[k_1]$ will be set to one at this point of the execution of $\mathcal{V}_{\text{new}}$. $\square$

Let us consider another modification of $\mathcal{V}_{\text{sun}}$, denoted by $\mathcal{V}''_{\text{sun}}$, which executes like $\mathcal{V}_{\text{sun}}$, except that when choosing a new block to verify, $\mathcal{V}''_{\text{sun}}$ does not limits its choice to the blocks marked as changed, but chooses *any* possible block (obviously, $\mathcal{V}''_{\text{sun}}$ does not necessarily terminate).

Lemma 3. *Let $\mathbb{P}$ be a program accepted by $\mathcal{V}_{sun}$. For all admissible executions $(i_1, \ldots, i_k, \ldots)$ for $(\mathbb{P}, \mathcal{V}_{new})$, there exists an admissible execution for $(\mathbb{P}, \mathcal{V}''_{sun})$, $(j_1, \ldots, j_l)$, such that*

$$\underline{F}_{i_1,\ldots,i_k}(\mathcal{V}_{new}) \preceq F_{j_1,\ldots,j_l}(\mathcal{V}''_{sun}) \ .$$

Proof. We proceed by induction on k. Let us assume that $\mathcal{V}_{\text{new}}$ has run through $(i_1, \ldots, i_k)$, and let us consider an admissible execution $(j_1, \ldots, j_l)$ for $\mathcal{V}''_{\text{sun}}$ as stated. At that point, $\mathcal{V}_{\text{new}}$ sets to one several new $\texttt{changed\_frame}[k_i]$, and modifies the corresponding frames $\texttt{u\_in}[k_i]$, $i = 1, \ldots, m$. For each of these k_i, let us denote by p_i a path from $\mathbb{B}_{i_k}$ to $\mathbb{B}_{k_i}$. Then it is clear that for all admissible i_{k+1},

$$\underline{F}_{i_1,\ldots,i_k,i_{k+1}}(\mathcal{V}_{\text{new}}) \preceq F_{j_1,\ldots,j_l,i_k,p_1,\ldots,p_m}(\mathcal{V}''_{\text{sun}}) \ ,$$

which concludes the proof. $\square$

Proposition 4. *A program $\mathbb{P}$ is accepted by $\mathcal{V}_{sun}$ iff it is accepted by $\mathcal{V}_{new}$.*

Proof. It is clear that the stack verification and the 'fall off' test work identically for $\mathcal{V}_{\text{sun}}$ and $\mathcal{V}_{\text{new}}$. We thus concentrate only on the type inference verification.

Let $\mathbb{P}$ be a program rejected by $\mathcal{V}_{\text{sun}}$. Then from Lemma 1, $\mathbb{P}$ is rejected by $\mathcal{V}'_{\text{sun}}$. Let us denote by $(i_1, \ldots, i_k)$ the admissible execution for $(\mathbb{P}, \mathcal{V}'_{\text{sun}})$ which has led to the beginning of the verification of the block $\mathbb{B}_{i_{k+1}}$ where the failure occurred. From Lemma 2, $(i_1, \ldots, i_{k+1})$ is an admissible execution of $\mathcal{V}_{\text{new}}$. Furthermore, $(i_1, \ldots, i_k)$ is also an admissible execution of $\mathcal{V}_{\text{new}}$, and at this point, the frame-set in $\mathcal{V}_{\text{new}}$ is higher than the frame-set in $\mathcal{V}'_{\text{sun}}$. This implies that $\mathcal{V}_{\text{new}}$ will return a failure after verifying $\mathbb{B}_{i_{k+1}}$.

Conversely, let $\mathbb{P}$ be a program accepted by $\mathcal{V}_{\text{sun}}$. First, it is clear that $\mathcal{V}''_{\text{sun}}$ will never return a failure when verifying $\mathbb{P}$: $\mathcal{V}''_{\text{sun}}$ simply verifies additional blocks without modifying their frame-set. Let us assume that the verification of $\mathbb{P}$ by $\mathcal{V}_{\text{new}}$ returns a failure, and let us consider the execution path for $(\mathbb{P}, \mathcal{V}_{\text{new}})$, $(i_1, \ldots, i_k)$, which has led to the beginning of the verification of the block i_{k+1} where the failure occurred. Let us consider an admissible execution for $\mathcal{V}''_{\text{sun}}$, as in Lemma 3, denoted by $(j_1, \ldots, j_l)$. Now, we arbitrarily force $\mathcal{V}''_{\text{sun}}$ to start verifying the block i_{k+1}. At this point, the frame-set in $\mathcal{V}''_{\text{sun}}$ is higher than the frame-set in $\mathcal{V}_{\text{new}}$, which implies that $\mathcal{V}''_{\text{sun}}$ should return a failure after having verified block $\mathbb{B}_{i_{k+1}}$, which contradicts our assumption. □

6 Practical Benchmarks

Although we did not implement a complete memory-constrained verifier, we wrote a simple software that builds the used-frames for a given `*.jca` file and counts the number of RAM cells necessary to verify its most greedy method.

We added two further optimizations to our software:

– In many cases the types used and produced by a byte-code are *unique* and *fully determined* by the byte-code itself. Typically, an `iload` requires the n-th local variable to be `int` and `int` only. Local variable n can hence be omitted from the frame whenever it is being read by an `iload` before being used by any other byte-code. Indeed, unification of this variable (within this block) is useless, given that its type can be nothing but `int`. A similar optimization applies to stack elements.
– A second optimization consists in identifying in each frame the variables and stack elements which are overwritten before even being read. This means that the old values (and types) of these variables are discarded. Hence, there is no need to keep track of such variables in the corresponding block-frames.

We used for our benchmarks the representative Java card applets from [10]. Results are rather encouraging, the new verification strategy seems to save on the average 80% of the memory claimed by [6]. Increase in workload (*i.e.*, a number of extra unifications) has not been explored as yet.

Applet	Sun's Verifier [6]	Memory-Constrained Verification	gain
NullApp.jca	6 words	3 words	50%
HelloWorld.jca	40 words	6 words	85%
JavaLoyalty.jca	48 words	13 words	73%
Wallet.jca	99 words	16 words	84%
JavaPurse.jca	480 words	41 words	91%
Purse.jca	550 words	39 words	93%
CryptoApplet.jca	4237 words	281 words	93%

References

1. A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.
2. P. Cousot, R. Cousot, *Abstract Interpretation: a Unified Lattice Model for Static Analysis by Construction or Approximation of Fixpoints*, Proceedings of POPL'77, ACM Press, Los Angeles, California, pp. 238-252.
3. X. Leroy, *Java Byte-Code Verification: an Overview*, In G. Berry, H. Comon, and A. Finkel, editors, Computer Aided Verification, CAV 2001, volume 2102 of Lecture Notes in Computer Science, pp. 265-285, Springer-Verlag, 2001.
4. X. Leroy, *On-Card Byte-code Verification for Java card*, In I. Attali and T. Jensen, editors, Smart Card Programming and Security, proceedings E-Smart 2001, volume 2140 of Lecture Notes in Computer Science, pp. 150-164, Springer-Verlag, 2001.
5. X. Leroy, *Bytecode Verification for Java smart card*, Software Practice & Experience, 32:319-340, 2002.
6. T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, The Java Series, Addison-Wesley, 1999.
7. S. Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann, 1997.
8. G. Necula, *Proof-carrying code*, Proceedings of POPL'97, pp. 106-119, ACM Press, 1997.
9. D. Schmidt, *Denotational Semantics, a Methodology for Language Development*, Allyn and Bacon, Boston, 1986.
10. P. Bieber, J. Cazin, A. El-Marouani, P. Girard, J.-L. Lanet, V. Wiels, G. Zanon, *The PACAP prototype: a tool for detecting java card illegal flows*, In I. Attali and T. Jensen, editors, Java on Smart Cards: Programming and Security, vol. 2041 of Lecture Notes in Computer Science, pp. 25-37, Springer-Verlag, 2001.

The Risks of Compromising Secret Information

Kyungah Shim

KISA(Korea Information Security Agency)
78, Garak-Dong, Songpa-Gu, Seoul 138-803, Korea
kashim@kisa.or.kr

Abstract. This paper presents a taxonomy of known-key attacks on athenticated key agreement protocols, which is based on the adversaries' roles and objectives. The taxonomy is illustrated using new attacks on the Just-Vaudenay and Yacobi protocol. The taxonomy is used to discuss countermeasures and protocol analysis methods against known-key attacks.

Keywords: Authenticated key agreement protocol, insider, known-key attack, key-compromise impersonation attack.

1 Introduction

Key establishment is the process by which two or more entities establish a shared secret key. The key is subsequently used to achieve some cryptographic goals such as confidentiality or data integrity. Broadly speaking, there are two kinds of key establishment protocols : i) key transport protocols in which a key is created by one entity and securely transmitted to the second entity, and ii) key agreement protocols in which both parties contribute information which jointly establish the shared secret key. In this paper, we shall focus on key agreement protocols for the asymmetric two-entity setting.

Let A and B be two honest entities, i.e., legitimate entities who execute the steps of a protocol correctly. A key agreement protocol is said to provide *implicit key authentication* of B to A if entity A is assured that no other entity aside from a specifically identified second entity B can possibly learn the value of a particular secret key. Note that the property of implicit key authentication does not necessarily mean that A is assured of B actually possessing the key. A key agreement protocol which provides implicit key authentication to both participating entities is called an *authenticated key agreement* (AK) protocol.

A key agreement protocol is said to provide *key confirmation* if entity A assured that the second entity B actually has possession of a particular secret key. If both implicit key authentication and key confirmation of B to A are provided, then the key establishment protocol is said to provide *explicit key authentication* of B to A. A key agreement protocol which provides explicit key authentication to both participating entities is called an *authenticated key agreement with key confirmation* (AKC) protocol.

Since the basic Diffie-Hellman key agreement scheme which provides the first practical solution to the key distribution problem, allowing two parties, never having met in advance or shared keying material to establish a shared secret by exchanging messages over an open channel was proposed, numerous protocols have been proposed to meet a variety of desirable security and performance requirements. In fact, the protocol design is error-prone. Many of these protocols were subsequently found to be flawed, and then either were modified to resist new attacks. In fact, it is known that Unified Model, KEA, MTI/C0 and MQV protocol are vulnerable to a key-compromise impersonation attack, a key recovery attack, a small subgroup attack, and an unknown key-share attack, respectively [3,11,10,8,12]. The robustness of a protocol against the compromise of secret information (past session keys or long-term private keys) has been the subject of many investigations [4,6,5,13,16]. The assumption with disclosure of some secret information may be theoretical, but, a good protocol minimizes the damage caused by secret information exposures. In this paper, we present a taxonomy of known-key attacks on the AK protocols, which is based on the adversaries' roles and objectives. The taxonomy is complete in the sense that all known attacks can be classified as falling into one of the categories. Also, the attacks caused by compromising of secret information is illustrated using new attacks on the AK protocols.

The remainder of the paper is organized as follows. In section 2, we briefly explain adversaries, adversaries' goals and the desirable security attributes of AK protocols to present a taxonomy of known-key attacks. In section 3, we present a taxonomy of known-key attacks. In section 4, we show how they can be applied to the Just-Vaudenay and Yacobi protocol. Also, we show that the Just-Vaudenay protocol (and the Song-Kim protocol) is vulnerable to a key-compromise impersonation attack and make suggestions for improvement. The concluding remark will be followed in section 5.

2 Adversaries and Desirable Security Attributes of AK Protocols

Before presenting a taxonomy of known-key attacks, we first describe the types of adversaries which is based on the adversaries' roles ;

- a passive adversary : an adversary who is capable only of recording (eavesdropping) the protocol runs,
- an active adversary : an adversary who may also transmit, alter, delete, inject data and interleave multiple instantiations of the same protocol.

A secure protocol should be able to withstand both passive and active adversary. The following kinds of adversaries is based on the type of information available to them [14] ;

- an outsider : an adversary with no special knowledge beyond that generally available, e.g., by eavesdropping on protocol messages over open channels,
- an insider : an adversary with access to additional information (e.g., past session keys, long-term private keys or secret partial information) obtained by some privileged means (e.g., physical access to private computer, conspiracy, etc.).

A good protocol should prevent an insider as well as an outsider. Next, we describes the adversaries' goals. An adversary in the AK protocols may pursue many strategies, including attempting to ;

- deduce session keys
  - using information gained by eavesdropping.
  - participate covertly in a protocol initiated by one entity with another, and influence it, by altering messages.
- impersonation : without being able to deduce the session key, deceive a legitimate entity regarding the identity of the entity with which is shares a key.
- deduce session key and impersonation : initiate one or more protocol runs (possibly simultaneously), and combine (interleave) messages from one with another, so as to masquerade as some party or carry out one of the above attacks and to deduce session key.
- deduce secret partial information of a legitimate entity : deduce a long-term private key of a legitimate entity without impersonation. (e.g., long-term private key of a legitimate entity).
- to exhaust the responder's resources and to disturb executions of it between honest initiator and the responder (denial-of-service attacks on the AK protocols).

In fact, understanding an adversary's possible attack objectives and roles, together with the techniques an adversary can use to launch attacks, can help sufficiently identifying the weakness of a given protocol as well as possible attacks.

In addition to implicit key authentication and key confirmation, a number of desirable *security attributes* of AK and AKC protocols have been identified. Typically the importance of supplying these attributes will depend on the application.

1. *Known-key security.* Each run of a key agreement between A and B should produce a unique secret key : such keys are called *session* keys. A protocol should still achieve its goal in the face of an adversary who has learned some other session keys.
2. *Forward secrecy.* If long-term private keys of one or more entities are compromised, the secrecy of previous session keys established by honest entities is not affected.
3. *Key-compromise impersonation attribute.* Suppose A's long-term private key is disclosed. Clearly an adversary that knows this value can now impersonate A, since it is precisely this value that identifies A. However, it may be desirable in some circumstances that this loss does not enable the adversary to impersonate other entities to A.
4. *Unknown key-share attribute.* Entity B cannot be coerced into sharing a key with entity A without B's knowledge, i.e., when B believes the key is shared with some entity $C \neq A$, and A (correctly) believes the key is shared with B.

These are typically properties possessed by face-to-face key establishment protocol is used to provide security in real-life applications. We are interested in the known-key security and the key-compromise impersonation attribute which are related to compromise of secret information.

3 A Taxonomy of Known-Key Attacks

In this section, we present a taxonomy of known-key attacks which is based on the adversaries' roles, objectives and the types of compromising session keys. The following types of secret information is available by an insider ;

- one or more past session keys,
  - session keys of the session established between two legitimate entities A and B.
  - session keys of the session established between an interleaved adversary and a legitimate entity A or B.
- long-term private keys of one or more entities,
- past session keys and a long-term private key.

Earlier attacks on key distribution protocols such as man-in-the-middle attacks [7] and interleaving attacks [2], fail to take advantage of known-key attacks. Typically, they leads to disclosure of secret information which subsequently can be used to obtain session keys or to impersonate. Burmester [4] presented a different attack on the Yacobi protocol. The adversary in his attack is an insider who succeeds in computing the session key of earlier communication from subsequent sessions with the entities involved in the communication. We will categorize known-key attacks so that these all known-key attacks can be classified as falling into one of the categories.

The taxonomy splits attacks into Known-key passive attacks and Known-key active attacks, which is based on adversaries' roles. The Known-key active attacks category is further divided into two subcategories : Known-key active attacks without impersonation (an intruder's goal is only session key retrieval) and Known-key impersonation attacks (an adversary's goal is retrieval of the session key established with a legitimate entity as well as impersonation), which is based on the adversaries' goals. Known-key impersonation attacks subcategory consists of two classes : KKI 1 and KKI 2 which is based on the types of compromising session keys. The full taxonomy of known-key attacks is as follows.

A Taxonomy of Known-Key Attacks

1. **Known-key passive (KKP) attacks** : an adversary obtains some session keys used previously and then uses this information to determine new session keys.
2. **Known-key active (KKA) attacks** : an adversary obtains some keys of the session established between interleaved adversary and a legitimate entity

A or B and then uses this information to determine new session keys or past session key between A and B, and to masquerade as some party, i.e., the adversary's goals are impersonation or session key retrieval.

(a) **Known-key active attacks without impersonation** : an adversary obtains some keys of the session established between interleaved adversary and a legitimate entity A or B and then uses this information to determine new session keys or past session key between A and B, i.e., the adversary's goal is only session key retrieval.

(b) **Known-key impersonation (KKI) attacks** : an adversary enters the session and impersonate himself/herself as a valid entity B with the previous session key and present key token, then finally compute the session key between A and B, i.e., the adversary's goal is impersonation as well as session key retrieval.

   i. **KKI 1 attacks** : session keys revealed to the adversary are the key of the session established between interleaved adversary and a legitimate entity A or B.

   ii. **KKI 2 attacks** : session keys revealed to the adversary are the key of the session established between two legitimate entities A and B as well as that of the adversary's interleaved sessions.

4 Examples of Attacks Caused by Compromising of Secret Information

In this section, we show that the Just-Vaudenay protocol is vulnerable to two active attacks caused by compromising of secret information. Before describing the attacks on the protocol, we first introduce terminology used through this paper. This paper deals with the case where G is a prime order subgroup of $\mathbb{Z}_p^*$, the multiplicative group of the integers modulo a prime p. The operation mod p will henceforth be omitted. However, the discussion applies equally well to any group of prime order in which the discrete logarithm problem is computationally intractable, for example, prime order subgroups of the group of points on an elliptic curve over a finite field.

• A, B	Honest entities.
• p	1024 bits primes.
• q	160 bits prime divisor of $p-1$.
• g	An element of order q in $\mathbb{Z}_p^*$.
• $a,\ b$	Static private keys of A and B : $a,\ b \in_R \mathbb{Z}_q^*$.
• $Y_A,\ Y_B$	Static public keys of A and B : $Y_A = g^a$, $Y_B = g^b$.
• $x,\ y$	Ephemeral private keys of A and B : $x,\ y \in_R \mathbb{Z}_q^*$.
• $R_A,\ R_B$	Ephemeral public keys of A and B : $R_A = g^x$, $R_B = g^y$.

The domain parameters (p, q, g) are common to all entities. For the remainder of this paper, we will assume that static public keys are exchanged via certificates. The certificate is given by

$$Cert_A =< ID_A, p, q, g, Y_A = g^a, Sig_{CA}(ID_A, p, q, g, Y_A) > .$$

$Cert_A$ denotes A's public-key certificate, containing a string of information that uniquely identifies A (such as A's name and address), her static public key Y_A, and a certifying authority CA's signature over this information. Other information may be included in the data portion of the certificate, including the domain parameters if these are not known from context. Any other entity B can use his authentic copy of the CA's public key to verify A's certificate, thereby obtaining an authentic copy of A's static public key.

4.1 KKI 1 Attack on the Just-Vaudenay Protocol IIA

At Asiacrypt'96, Just and Vaudenay [9] proposed a two-party authenticated key agreement protocol. And they showed that impersonating either A or B in the protocol was equivalent to solving an instance of the Diffie-Hellman problem. However, their proof does not imply that the protocol is secure against active attacks. The protocol is shown in Fig. 1 (with the session key $K = g^{ay+xy+bx}$).

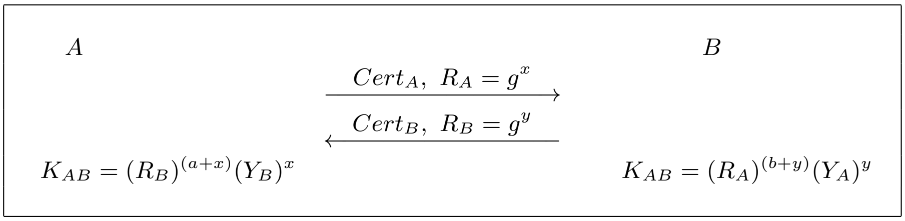

A
B
$Cert_A,\ R_A = g^x$
$Cert_B,\ R_B = g^y$
$K_{AB} = (R_B)^{(a+x)}(Y_B)^x$
$K_{AB} = (R_A)^{(b+y)}(Y_A)^y$

Fig. 1. The Just-Vaudenay protocol IIA

In the KKI 1 attack on the protocol, an adversary succeeds in computing the new session keys from compromising key of an earlier session with the entity being deceived as well as impersonation. The KKI 1 attack is executed as follows. When the entity A sends $R_A = g^x$ to B, an adversary E intercepts it. E computes $R_B = g^y$ as follows.

1. E chooses a random value $t \in \mathbb{Z}_q^*$. And let t be $b + y$.
2. E computes g^t.
3. E obtains g^y by computing $(g^t)(g^b)^{-1}$ using B's public key $Y_B = g^b$.

Note that neither b nor y is known to E. Next, E (pretending to be B) sends $R_B = g^y$ to A. Then A computes the session key $K_{AB} = (R_B)^{(a+x)}(Y_B)^x = g^{xy+ay+bx}$, but E cannot compute it. Suppose that the session key K_{AB} is later revealed to E. Then E can obtain g^{ay} by computing

$$K_{AB}/(R_A)^t = g^{xy+bx+ay}/(g^x)^{(b+y)} = g^{ay}.$$

Then E can impersonate B to A and computes all the session keys for next sessions. Indeed, E (pretending to be B) starts the protocol with A and replays the message $R_B = g^y$. Then A sends $R_A = g^{x'}$. Then E can compute the session key K from the values t and g^{ay} as follows :

$$K = (R_A)^t g^{ay} = g^{x'y+x'b+ay}.$$

A also computes the same session key $K = (R_B)^{(a+x')}(Y_B)^{x'} = g^{ay+x'y+bx'}$. Thus, E succeeds to impersonate B to A and obtains the session key K.

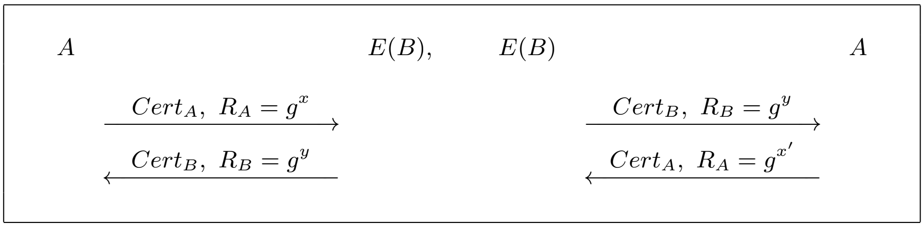


Fig. 2. KKI 1 attack on the Just-Vaudenay protocol

Remark. 1. In fact, E can induce A to reveal the key, K_{AB} established in the sessions between A and B. This may be reasonable assumption since A believes that the session key should be known to B. The attack scenario is described in [4].
2. By definitions, AK protocols satisfies the implicit key authentication (IKA) but the property of IKA does not necessarily mean that A is assured of B actually possessing the key. In fact, keys established using AK protocols should be confirmed prior to cryptographic use. Indeed, some standards such as ANSI X9.42 take the conservative approach of maintaining key confirmation of keys agreed in an AK protocol. Thus, after finishing the session established between A and E (pretending to be B), if appropriate key confirmation is subsequently provided, then A comes to know the intrusion of the adversary in that session. Thus the session key K_{AB} of A may be handled with careless and then be revealed. For, A thinks that it is useless and meaningless.

Another risk. The adversary E can obtain the value g^{ab} in the above attack. When the session key K_{AB} is revealed to E, E can obtain the value g^{ab} as follows;

1. First, E computes $K' = (R_A \cdot Y_A)^t = g^{(x+a)(y+b)} = g^{xy+ay+bx+ab}$.
2. Then, E can obtain the value g^{ab} by computing K'/K_{AB}.

In fact, the value g^{ab} is the resulting shared secret of the static Diffie-Hellman scheme. In Fig. 3, if A (resp., B) a priori has an authentic copy of B's (resp., A's) static public key, this scheme can be executed without interaction. Given that no parties are on-line, the static Diffie-Hellman scheme is the only choice. The scheme provides only mutual implicit key authentication, no other security attributes are provided. As the resulting shared secret value will always consist of the same value : it is strongly suggested that additional public information always be used with varying values in the key derivation function. Thus, the disclosure of the value g^{ab} brings out the serious consequence in these environment.

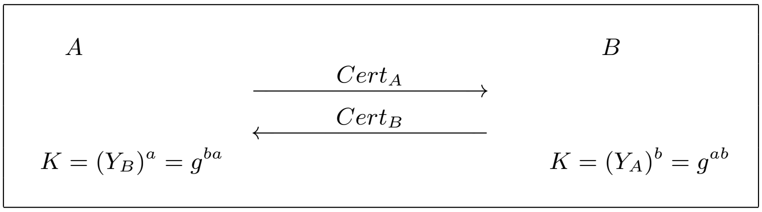


Fig. 3. The static Diffie-Hellman protocol

Remark. This KKI 1 attack cannot be applied to the Yacobi protocol and Goss protocol. In the Yacobi protocol, when A sends $R_A = g^x$ to B, the adversary E intercepts it. E chooses a random number y and sends $R_B = g^y$ to A pretending to be B. Then A computes the session key $K_{AB} = g^{ay+bx}$. But, E cannot compute the session key since she does not know the long-term private key of B. Even though it is revealed to E, E can recover only g^{bx}. However, this value depends on the value R_A which is changed in each session. Thus, this value cannot help the adversary to obtain the new session keys.

4.2 KKA Attack without Impersonation on the Yacobi Protocol

We describe a KKA attack on the Yacobi protocol which is a variant of Burmester's triangle attack on the same protocol. In the Burmester's attack, the adversary is an insider who knows a long-term private key of A. However, in our attack, we replace the insider with a dishonest entity. Suppose that C is a dishonest entity (C need not to know the long-term private key of A) and C has her certificate $Cert_C$ with a public/private key pair ($Y_C = g^c,\ c$). The Yacobi protocol is shown in Fig. 4. Then the KKA attack without impersonation on the Yacobi protocol is launched as follows.

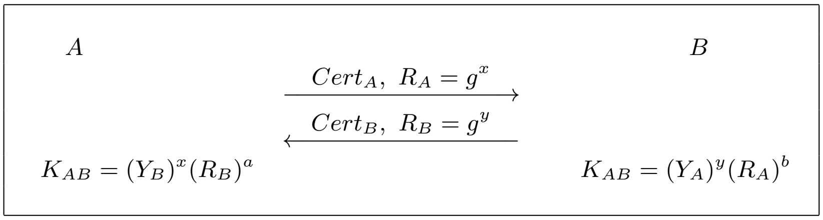


Fig. 4. The Yacobi protocol

1. First, the adversary E eavesdrops on a communication of A and B.
2. Subsequently, C communicates with A and B separately. When A sends $g^{x'}$ to C, C replays g^x as the second message to A. Also, C starts a communication with B replaying g^x, and B sends $g^{y'}$ to C. Then A can compute the session key $K_{AC} = (g^y)^a \cdot (g^c)^{x'}$ and B can compute the session key $K_{BC} = (g^x)^b \cdot (g^c)^{y'}$. However, C can obtain neither K_{AC} nor K_{BC}.
3. Suppose that the session keys K_{AC} and K_{BC} are later revealed to C. With these information and her own long-term private key c, C can recover the session key established in step 1 by computing

$$K_{AB} = K_{AC} \cdot K_{BC} \cdot ((g^{x'})^c)^{-1} \cdot ((g^{y'})^c)^{-1} = g^{ay+bx}.$$

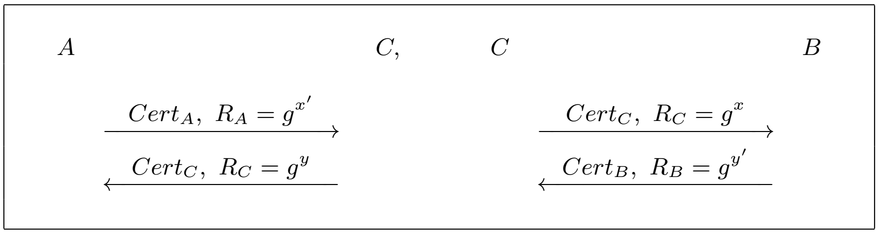


Fig. 5. KKA attack on the Yacobi protocol

Note that both this attack and the Burmester's triangle attack on the Yacobi protocol are falling into the KKA attacks without impersonation category in the taxonomy.

4.3 Key-Compromise Impersonation Attack on the Just-Vaudenay Protocol

Key-compromise impersonation resiliance is a security attribute that provides assurance to an entity A that, even if an adversary somehow obtain A's long-term private key, the adversary cannot successfully impersonate another entity to A. In fact, long-term secrets are in practice the most vulnerable secrets in the system : in a typical setting, they are stored on a disk, perhaps protected by a password. Ephemeral data is much more difficult for an attacker to obtain. Indeed, we note the compromise of long-term private keys does not necessarily mean that they are obtained via an inversion of the static public key. Since users must store their private keys for use in key computation, the private keys may also be obtained through lack of suitable physical measures. Not only this compromises the security and validity of any transmitted message issued after the break but it also compromises all past session keys. This property may be attractive for the robustness of the security in most commercial applications where customers does not always protect their key sufficiently.

The Just-Vaudenay protocol is also vulnerable to a key-compromise impersonation attack. Suppose that A's long-term private key a is compromised. And suppose that an adversary E who knows the value wants to impersonate B to A. First, E chooses a random value $t \in \mathbb{Z}_q^*$. Let t be equal to $b + y$. Note that neither b nor y is known to E. However, E can obtain g^y by computing $g^t(g^b)^{-1}$.

1. A selects $x \in_R \mathbb{Z}_q^*$ and sends $R_A = g^x$ and $Cert_A$ to B.
2. $E(B)$ (pretending to be B) sends $R_B = g^y$ and $Cert_B$ to A.
3. A computes a shared secret $K = (R_B)^{(a+x)}(Y_B)^x = g^{xy+ay+bx}$.
4. $E(B)$ computes the shared secret as follows

$$K = (R_A)^t(R_B)^a = g^{xy+xb+ay}.$$

The adversary would be able to generate a random value t, and removing the public key of corresponding private key, she can produce new transmission public key R_B. Finally, she succeeds to impersonate B to A and retrieves the shared secret K.

Remark. 1. At Indocrypt'00, Song and Kim [15] proposed an authenticated key agreement protocol which is an elliptic curve version of the Just-Vaudenay protocol. It is still vulnerable to the same key-compromise impersonation attack. However, their scheme is secure against the KKA attack described in the section 4.1. Because they use a key derivation function to calculate a session key from a shared secret.
2. By definitions, this attack is not included in the category of known-key attacks.

4.4 A Modified Version of the Just-Vaudenay Protocol

We propose a modified version of the Just-Vaudenay protocol to resist the attacks described in previous sections. We will use the idea used in the MQV protocol [11]. The following notation is used. If $X \in [1, p-1]$ then $\overline{X} = (X \bmod 2^{80}) + 2^{80}$. Note that $(\overline{X} \bmod q) \neq 0$.

Modification

1. A compute a shared secret $K_{AB} = (R_B)^{(a+x\overline{R_A}\cdot\overline{R_B})}\cdot(Y_B)^x = g^{ay+bx+xy\overline{R_A}\cdot\overline{R_B}}$.
2. B compute a shared secret $K_{AB} = (R_A)^{(b+y\overline{R_A}\cdot\overline{R_B})}\cdot(Y_A)^y = g^{ay+bx+xy\overline{R_A}\cdot\overline{R_B}}$.

In the modified version, we cannot determine t and the ephemeral public key $R_B = g^y$ such that $t = b+y\overline{R_A}\cdot\overline{R_B}$. Indeed, an adversary chooses a random value t and let be $t = b+y\overline{R_A}\cdot\overline{R_B}$. Next, she computes $g^{y\overline{R_B}} = (g^t(g^b)^{-1})^{\overline{R_A}^{-1} \ (mod\ q)}$. But, she does not obtain g^y since she cannot determine the value R_B in advance. Thus, adding $\overline{R_A}\cdot\overline{R_B}$ in the exponent of resulting shared key can prevent the KKI 1 attack and the key-compromise impersonation attack described in previous sections.

Remark. In fact, the use of key derivation function to derive a session key from the shared secret can help prevent against some kinds of known key attacks destroying the algebraic relationships between the shared secret K and the static and ephemeral public keys. But, without using of such an additional function, the protocol should be designed so that all the session keys of the same protocol should be independent.

5 Conclusion

We have described a taxonomy of known-key attacks on the authenticated key agreement protocols. The taxonomy based on adversaries' roles and objectives is helpful for cryptographic protocol analysis including formal methods. Also, we have shown how the attacks can be applied to the Just-Vaudenay protocol and Yacobi protocol. Moreover, we shown that the Just-Vaudenay protocol is vulnerable to a key-compromise impersonation attack and make suggestions for improvement.

Anderson and Needham [1] presented "Assume nothing : Do not assume the secrecy of anybody else's secrets" principle for designing cryptographic protocols. In fact, the adversary who break cryptographic protocols don't follow rules : they cheat. They can attack a cryptographic protocol using techniques the designers never thought of. They steal technical data, bribe the entities, and collude. Defenders have to protect against every possible vulnerabilities, but an adversary only has to find one security flaw to compromise the protocol. Actually, the long-term secret of each entity may be compromised. Although precautions may be taken to avoid compromising of session keys, an adversary might obtain one of those keys. Also, keying material cannot be trusted to provide the required security : confidentiality, integrity, association with the owner or other entity and association with other information. In particular, certificates for keys should expire : and when one key is expiring, it should not be used for encrypting the new key that will replace it. It is vital that private keys are not leaked to attackers, and there may need to be protection against insiders as well as outsiders.

In fact, the attacks described in this paper may be theoretical. Also, well-designed implementations of key distribute systems will prevent session keys being disclosed or lost. In real systems, the users do not know their encrytion keys, and only interact through well defined interfaces. However, one should be worry, particularly with poor implementations, or with applications in which the session keys are eventually disclosed. A good protocol design will minimize the effects of such events. Therefore, one should design these protocols with the assumption on malicious adversaries intent on defeating the protocol is living inside the system and so they use secret information at will.

References

1. R. Anderson and R. Needham, Robustness Principles for Public Key Protocols, Advances in Cryptology, Crypto '95, LNCS 963, Springer-Verlag, pp. 236-247, 1995.
2. R. Bird, I. Gopal, P. A. Janson, S. Kutten, R. Mulva, and M. Yung, Systemetic design of two-party authentication protocols, Proc. of Crypto'91: Advaced in cryptology LNCS 576, Springer-Verlag, pp. 44-61, 1991.

3. S. Blake-Wilson and A. Menezes, Authenticated Diffie-Hellman key agreement protocols, Proc. of the 5th Annual Workshop on Selected Areas in Cryptography (SAC'98), LNCS 1556, Springer-Verlag, pp. 339-361, 1999.
4. M. Bumester, On the risk of opening distributed keys, Advances in Cryptology, Crypto'94, LNCS 839, pp. 308-317, 1994.
5. D. Denning, and G. Sacco, 'Timestamps in key distribution systems', Communications of the ACM, 24(8), pp. 533-536, 1981.
6. Y. Desmedt and M. Bumester, Towards practical 'proven secure' authenticated key distribution, ist ACM Conference on Computer and Communications security, ACM press, pp. 228-231, 1993.
7. W. Diffe, P. van Oorschot and M. Wiener, Authentication and authenticated key exchange, Designs, Codes and Cryptography, 2, pp. 107-125, 1992.
8. K. C. Goss, Cryptographic method and apparatus for public key exchanges with authentication, U. S Patent # 4,956,863, 11, Sep., 1990.
9. M. Just and S. Vaudenay, Authenticated multi-party key agreement, Advances in Cryptology, Asiacrypt'96, LNCS 537, pp. , 19.
10. B. Kaliski, Contribution to ANSI X9F1 and IEEE P1363 working groups, June, 1998.
11. L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone, An efficient protocol for authenticated key agreement, Technical report CORR 98-05, University of Waterloo, 1998.
12. C. Lim and P. Lee, A key recovery attack on discret log-based schemes using a prime order subgroup, Advaced in cryptology, Crypto97', LNCS 1294, Springer-Verlag, pp. 249-263.
13. T, Mastumoto, Y. Takashima, and H. Imai, On seeking smart public-key distribution systems, IEICE TRANS. FUNDAMENTALS, Vol. E69 pp. 99-106, 1986.
14. A. Menezes, P. van Oorschot and S. Vanstone, Handbook of Applied cryptology, CRC Press, 1997.
15. B. Song and K. Kim, Two-pass authenticated key agreemnet protocol with key confirmation, Progress in Cryptology, Indocrypto'00, LNCS 1977, pp. 237-249, 2000.
16. Y. Yacobi, A Key distribution "paradox", Advances in Cryptology, Crypto'89, LNCS 537, pp. 268-273, 1991.

Password-Authenticated Key Exchange between Clients with Different Passwords

Jin Wook Byun[1], Ik Rae Jeong[1], Dong Hoon Lee[1], and Chang-Seop Park[2]

[1] Center for Information Security Technologies(CIST)
Korea University, Anam Dong, Sungbuk Gu, Seoul, Korea
{byunstar,jir}@cist.korea.ac.kr, donghlee@korea.ac.kr
[2] Division of Computer Science & Electronics
Dankook University, Cheonan, Korea
csp0@dankook.ac.kr

Abstract. Most password-authenticated key exchange schemes in the literature provide an authenticated key exchange between a client and a server based on a pre-shared password. With a rapid change in modern communication environments, it is necessary to construct a secure end-to-end channel between clients, which is a quite different paradigm from the existing ones. In this paper we propose a new framework which provides a password-authenticated key exchange between clients based only on their two different passwords without any pre-shared secret, so called *Client-to-Client Password-Authenticated Key Exchange* (C2C-PAKE). Security notions and types of possible attacks are newly defined according to the new framework. We prove our scheme is secure against all types of attacks considered in the paper. Two secure C2C-PAKE schemes are suggested, one in a cross-realm setting and the other in a single-server setting.

Keywords: Password authentication, key exchange, cross-realm, *Kerberos*, dictionary attack.

1 Introduction

Authentication relying on passwords is a popular method for user authentication in the client-server model because of its easy-to-memorize property. There is, however, several security concerns such that a password selected from a small space allows an adversary to mount, off-line, a dictionary attack. To prevent this everpresent attack, various protocols have been proposed to achieve *secure password-authenticated key exchange* [2,12,15,1,4,11,8] based on different cryptographic assumptions.

Password-authenticated key exchange schemes assume that two entities have a priori shared password. Two parties use their shared password to generate a secure common session key and perform key confirmation with regard to the session key. Most password-authenticated key exchange schemes in the literature consider authentication between a client and a sever.

With diversity and development of communication environments in the fields such as mobile networks, home networking and etc., the end-to-end security is considered as one of main concerns [17,3]. For example, from a user's point of view, in a mobile computing environment, a secure end-to-end channel between one mobile user in *cell A* and another user in *cell A* or *cell B* may be a primary concern. Additionally, the end-to-end security service minimizes the interferences from the operator controlled network components.

The primary goal of the paper is to design a secure authenticated key exchange scheme, so called *Client-to-Client Password-Authenticated Key Exchange* (C2C-PAKE) between two clients relying only on their distinct passwords without any priori shared secret. Security notions and types of possible attacks are newly defined according to the new framework. We prove our scheme is secure against all types of attacks considered in the paper. Two secure C2C-PAKE schemes are suggested, one for a cross-realm setting where two clients are in two different realms and hence there exist two servers involved, the other for a single-server setting where two clients are in the same realm. Our schemes are based on an efficient cross-realm authentication scheme [5] in *Kerberos* system.

1.1 Related Work and Our Contribution

We explain our contributions in aspects of *Practical Use* and security against *Off-line Dictionary Attack* by comparing our C2C-PAKE with related works.

Practical Use. Steiner, Tsudik and Waider proposed an authenticated key exchange scheme, called 3-Party EKE, between two clients, extended from M-EKE [13]. But in their scheme, a server (KDC) must participate in every key exchange process between clients. This on-line intervention of the server in authenticated key exchange could be a critical bottleneck in efficiency of open networks. The proposed C2C-PAKE schemes reduce the on-line intervention of servers using *Ticket* which could be reused within its lifetime L between KDC and a client. Furthermore 3-Party EKE does not provide a cross-realm authentication while our C2C-PAKE scheme does.

Off-Line Dictionary Attack. One of the most serious problems in the *Kerberos* system is an off-line dictionary attack. To prevent a dictionary attack, *Kerberos* V5 introduced so-called *preauthentication* which has the form of an encrypted timestamp. Unfortunately, *Kerberos* V5 does not provide a complete solution against an off-line dictionary attack [16]. To solve this off-line dictionary attack, Jaspan [9] proposed a variant of *Kerberos* system, denoted as PA-ENC-DH by applying the DH-EKE scheme to *Kerberos* authentication scheme between a client and a server (KDC). Jaspan could solve long-standing limitations of the *Kerberos* system by proposing PA-ENC-DH scheme: vulnerability to dictionary attacks, dependence on insecure time synchronization, and vulnerability to the password chaining problem. However PA-ENC-DH scheme only considered single-server setting. Their scheme cannot be easily lift up to a scheme in a cross-realm setting since security notions should be redefined according to the changed setting. Our scheme, which is a variant of cross-realm authentication in the *Kerberos* system, solves dictionary attack against two passwords in a cross-realm setting. In addition to this, C2C-PAKE also solves the problems of time synchronization and password chaining.

2 Preliminaries

In this section we introduce definitions and models needed in our schemes. First we classify password-authenticated key exchange schemes into two models.

1. [**Shared Password-Authentication Model**] This authentication model (SPA, for short) provides password-authenticated key exchange using a shared password between a client A and a server B. We assume A has a secret password and B has a corresponding password verifier in its database. A and B authenticate each other based on the shared password. Most password-based schemes in the literature are based on this model.
2. [**Different Password-Authentication Model**] This model (DPA, for short) provides password-authenticated key exchange between two clients using different passwords. We further classify settings of this model into a cross-realm setting and a single-server setting. In a cross-realm setting, the model consists of two clients A_1 and A_2, and two servers B_1 and B_2, where A_1 and A_2 are users of B_1 and B_2, respectively. In a single-server setting, the model consists of two clients A_1 and A_2, and a server B_1 where A_1 and A_2 are users of server B_1. Clients A_1 and A_2 authenticate each other with their different passwords with help of B_1 and B_2 in the cross-realm setting, and with help of B_1 in the single-server setting. Our schemes in this paper are based on this model.

In the DPA two passwords are involved in authentication process while only a single password is used in the SPA. Notions of security and attacks in the literature have been defined under the environments of the SPA. To discuss the security of the DPA, the related notions should be modified accordingly. Next section we treat this issue informally.

2.1 Definitions and Assumptions

First, we introduce a *perfect forward secrecy in the SPA* and then define a corresponding notion in the DPA.

Definition 1. *A protocol is said to have* ***perfect forward secrecy in the SPA*** *if compromise of a shared password does not compromise past session keys*

In the DPA, with respect to perfect forward secrecy, the case of compromise of two long-term passwords should be considered.

Definition 2. *A cryptographic primitive or protocol provides* ***perfect forward secrecy in the DPA*** *if the compromise of two passwords can not compromise past session keys.*

We can easily infer from Definition 1 and 2 above that if a cryptographic primitive or protocol satisfies the *perfect forward secrecy in the DPA*, then a *perfect forward secrecy in the SPA* is satisfied, while the converse is not true. Next we overview the definition of a *Denning-Sacco attack in the SPA* [6] and define a new notion denoted as a *Denning-Sacco attack in the DPA*. As defined below, a *Denning-Sacco attack in the DPA* possibly compromises several session keys simultaneously.

Definition 3. *A protocol is said to be vulnerable to a* ***Denning-Sacco attack in the SPA*** *if compromise of a common session key allows an attacker to mount a dictionary attack on the long-term secret value (or password) or to impersonate one of the parties.*

Definition 4. *A protocol is said to be vulnerable to a* ***Denning-Sacco attack in the DPA*** *if compromise of session keys allows an attacker to mount a dictionary attack on the long-term secret values (or passwords) or to impersonate one of the parties.*

Password-based key exchange protocols in the SPA must have a property that the shared password is protected against an off-line dictionary attack. The protocols in the DPA also must be strong against this attack. But the number of passwords to be protected against dictionary attacks in the DPA is more than one. Therefore we define a security notion against a dictionary attack.

Definition 5. *A protocol is said to be strong against a* ***dictionary attack in the DPA*** if the following conditions are satisfied. First, all passwords in the protocol must be strong against a dictionary attack. Second, even if attacker $\mathcal{A}$ is given one password, other passwords must be strong against a dictionary attack.

There exist many other active attacks in the SPA. Examples are replay attack, man-in-the-middle attack, on-line guessing attack, etc. Such attacks could be similarly defined in the DPA without modification. Next we discuss the security of protocol in DPA .

Security of Protocol in the DPA. Suppose that poly-time adversary $\mathcal{A}$ tries to break a given protocol by using well-known attack methods. In case of on-line attack, $\mathcal{A}$ tries all possible passwords in log-in stage. If a server accepts a password chosen by $\mathcal{A}$, the guess is correct. Otherwise $\mathcal{A}$ eliminates the guess from the password-dictionary. $\mathcal{A}$'s probability of success after R successive rejections is $1/(|D| - R)$ where $|D|$ is a dictionary size of passwords. Actually, on-line attack can not be avoided and hence the success probability of on-line attack may be considered as a lower bound of an advantage of any adversary. That is, a protocol is secure if the advantage of an adversary $\mathcal{A}$ in attacking the protocol is bounded above, as follows.

Table 1. Notation

Notation	***Meaning***
Alice, *Bob*	honest user or client.
$ID(A), ID(B)$	identities of *Alice* and *Bob*.
pwa, *pwb*	passwords memorized by *Alice* and *Bob*.
E_X	symmetric encryption with X.
sk	session key between *Alice* and *Bob*.
H_1, H_2, H_3, H_4, H_5	cryptographic hash functions (e.q, SHA-1).
$Ticket_B$	Kerberos ticket issued to user A for service from B.

$$Adv_{\mathcal{A}}^{DPA}(k) \leq \mathcal{O}(1/(|D| - R)) + \varepsilon(k),$$

for some negligible function $\varepsilon(\cdot)$, where R is the number of times rejected in log-in stage, and k is a security parameter. The second term in the right-hand side of the inequality is $\mathcal{A}$'s advantage of success in all well-known active attacks in the DPA, which is negligible.

Computational Assumption. Our schemes are based on numerical assumptions and computational assumptions. Let p, q be sufficiently large primes such that $q|p - 1$, and let G be a subgroup of Z_p^* of order q. During initialization step, a generator $g \in G$ and hash function$(H_1, H_2, H_3, H_4, H_5)$ are published. All protocols throughout the paper are based on the discrete logarithm assumption(DLA) and the Diffie-Hellman assumption(DHA).

We use the notations in Table 1 throughout our paper.

3 Kerberos System in a Cross-Realm Setting

Kerberos is an authentication system developed as a part of project *Athena* at MIT [14]. In the *Kerberos* system, the distribution of a session key between a client and an application server is accomplished by the intervention of a *Kerberos* server consisting of *Authentication Server* and *Ticket Granting Server*. A full-service *Kerberos* provides a mechanism to support cross-realm authentication. In this section we roughly overview a cross-realm authentication scheme in *Kerberos* system. And then we also overview an efficient cross-realm authentication scheme, *Fake Ticket Protocol* [5] which is a variant of the original *Kerberos* system. This scheme provides an efficient framework for cross-realm authentication and decreases communication complexity between several realms [5].

Cross-Realm Authentication in the Original Kerberos System.
First we overview a cross-realm authentication in *Kerberos* system. Cross-realm authentication allows a client *Alice* in one realm to access a server S in other realm. The details of cross-realm authentication are illustrated in *Protocol 1.* In Step (1) *Alice* requests her Authentication Server(AS) for Ticket Granting Ticket(TGT_A). On request from *Alice*, AS_A issues TGT_A to be used to access

Ticket Granting Server(TGS_A) in Step (2). In Step (3) *Alice* requests TGS_A to issue TGT_B to be used to communicate with remote TGS_B. After verifying TGT_A, TGS_A sends TGT_B encrypted with K_{TGS_A,TGS_B} to *Alice* in Step (4). *Alice* with TGT_B requests remote server S for real *Ticket*, and then TGS_B issues to *Alice* $Ticket_B$ to be used to communicate with S.

$Alice \rightarrow AS_A$	:	ID(A)	(1)
$AS_A \rightarrow Alice$	:	TGT_A, $E_{K_A}(K_{A,TGS_A}, T)$	(2)
$Alice \rightarrow TGS_A$	:	ID(A), ID(S), TGT_A, $E_{K_{A,TGS_A}}(T)$	(3)
$TGS_A \rightarrow Alice$	:	TGT_B, $E_{K_{A,TGS_A}}(T, K_{A,TGS_B})$	(4)
$Alice \rightarrow TGS_B$	:	ID(A), TGT_B, $E_{K_{A,TGS_B}}(T)$	(5)
$TGS_B \rightarrow Alice$	:	$Ticket_B$, $E_{K_{A,TGS_B}}(K_{A,B}, T)$	(6)
$Alice \rightarrow S$	:	ID(A), $Ticket_B$, $E_{K_{A,B}}(T)$	(7)

Protocol 1. Cross-Realm Authentication in Kerberos System. Each realm consists of Authentication Server(AS) and Ticket Granting Server(TGS). TGT_A is a ticket to be used by *Alice* to access TGS_A. K_A and K_B are symmetric keys shared between *Alice* and AS_A, Server S and AS_B, respectively. T is a timestamp used to prevent replay attacks. Notation $K_{X,Y}$ means a key shared between entity X and entity Y. $TGT_A = E_{K_{TGS_A,AS_A}}(ID(A), K_{A,TGS_A})$, $TGT_B = E_{K_{TGS_A,TGS_B}}(ID(A), K_{A,TGS_B})$, and $Ticket_B = E_{K_B}(ID(A), ID(S), K_{A,B})$.

Fake Ticket Protocol [5]

Crescenzo and Kornievskaia presented a cross-realm authentication scheme, called Fake Ticket Protocol [5] between two clients *Alice* and *Bob* in different realms. Their scheme is a variant of *Kerberos*. We briefly describe Fake Ticket Protocol as follows. At login stage, *Alice* obtains a TGT_A from KDC_A. When *Alice* with TGT_A makes a request for a service ticket from KDC_A, it generates a random session key $K_{A,B}$ and issues fake ticket($FTKT$) to *Alice* in Step (4). *Alice* forwards $FTKT$ to *Bob*. In Step (8) *Bob* with $FTKT$ requests a real service ticket TKT to KDC_B. On request from *Bob*, KDC_B issues a real ticket(TKT) to be used to communicate with *Alice*. The rest part is similar to those of *Kerberos* cross-realm authentication.

We assume *Internet* is used between realms and *Intranet* used inside a realm. It is notable the amount of communication between realms is relatively small since *Internet* is slower than *Intranet*. As shown in *Protocol 2* the amount of communication between KDC_A and KDC_B in Fake Ticket Protocol is reduced to one while the original *Kerberos* system requires three communications between realms.

4 C2C-PAKE in a Cross-Realm Setting

In this section we propose a new client to client password-authenticated key exchange(C2C-PAKE) scheme in a cross-realm setting. C2C-PAKE is con-

I. Alice's Login Stage in KDC_A

$Alice \rightarrow KDC_A$	:	ID(A)	(1)
$KDC_A \rightarrow Alice$	:	TGT_A, $E_{K_A}(K_{A,TGS_A}, T)$	(2)

II. FTKT Issuing Stage

$Alice \rightarrow KDC_A$	:	ID(A), ID(B), TGT_A, $E_{K_{A,TGS_A}}(T)$	(3)
$KDC_A \rightarrow Alice$	:	$FTKT$, $E_{K_{A,TGS_A}}(T, K_{A,B})$	(4)
$Alice \rightarrow Bob$	:	ID(A), $FTKT$, $E_{K_{A,B}}(T)$	(5)

III. Bob's Login Stage in KDC_B

$Bob \rightarrow KDC_B$	:	ID(B)	(6)
$KDC_B \rightarrow Bob$	:	TGT_B, $E_{K_B}(K_{B,TGS_B}, T)$	(7)

IV. Real Ticket Issuing Stage

$Bob \rightarrow KDC_B$	:	$FTKT$, TGT_B, $E_{K_{B,TGS_B}}(T)$	(8)
$KDC_B \rightarrow Bob$	:	TKT, $E_{K_{B,TGS_B}}(T)$	(9)

Protocol 2. Fake Ticket Protocol. $FTKT = E_{K_{KDC_A,KDC_B}}(ID(A), K_{A,B})$, $TKT = E_{K_B}(ID(A), ID(B), K_{A,B})$, $TGT_A = E_{K_{TGS_A,AS_A}}(ID(A), K_{A,TGS_A})$ and $TGT_B = E_{K_{TGS_B,AS_B}}(ID(B), K_{B,TGS_B})$. Notation $K_{X,Y}$ means a key shared between entity X and entity Y.

structed based on the framework of Fake Ticket Protocol [5] described in the previous section. In a distributed environment, it is inefficient that KDC_A always participates in every process of session key generation between two clients *Alice* and *Bob* in two different realms as in Fake Ticket Protocol. Our scheme uses *Ticket* structure. By using $Ticket_B$, *Alice* can generate a session key only using her memorable passwords without intervention of KDC_A.

C2C-PAKE described in Protocol 3 consists of two stages. One is a ticket issuing stage((1)-(2)), and the other is a mutual authentication and session key generation stage((3)-(8)). First, KDC_A issues a $Ticket_B$ in *Ticket Issuing Stage*. As $Ticket_B$ is issued in advance and contains lifetime, KDC_A does not need to participate in every session key generation. *Alice* can re-use this $Ticket_B$ to generate a session key with *Bob* whenever needed during the lifetime L specified in the $Ticket_B$.

Protocol Initialization: Preliminaries for a protocol run are as follows.

1. g, p and q are global parameters shared by protocol participants.
2. *Alice* chooses a password pwa, then transfers it to KDC_A through a secure channel. *Bob* also transfers pwb to KDC_B similarly. KDC_A and KDC_B store ($ID(A)$, pwa) and ($ID(B)$, pwb) respectively in their own databases.

Protocol Description: We assume that there is a secure shared key K between KDC_A and KDC_B by using any of the methods in the public-key cryptography. An example of such a method can be found in PKCROSS [10].

1. *Alice* chooses $x \in Z_p^*$ randomly, computes and sends $E_{pwa}(g^x)$ to KDC_A together with ID(A) and ID(B).

I. Ticket Issuing Stage			
Alice $\rightarrow KDC_A$	:	$ID(A)$, $ID(B)$, $E_{pwa}(g^x)$	(1)
$KDC_A \rightarrow$ *Alice*	:	$E_R(g^x \oplus g^r, ID(A), ID(B))$, $E_{pwa}(g^y)$, $Ticket_B$	(2)
II. Mutual Authentication and Session Key Generation			
Alice $\rightarrow$ *Bob*	:	$Ticket_B$, ID(A), L	(3)
Bob $\rightarrow KDC_B$	:	$Ticket_B$, $E_{pwb}(g^{x'})$, $ID(A)$, $ID(B)$, L	(4)
$KDC_B \rightarrow$ *Bob*	:	$E_{R'}(g^{pwa \cdot r \cdot r'} \oplus g^{x'}, ID(A), ID(B))$, $E_{pwb}(g^{y'})$ $E_{H_4(g^{pwa \cdot r})}(g^{pwb \cdot r \cdot r'})$	(5)
Bob $\rightarrow$ *Alice*	:	$E_{cs}(g^a)$, $E_{H_4(g^{pwa \cdot r})}(g^{pwb \cdot r \cdot r'})$	(6)
Alice $\rightarrow$ *Bob*	:	$E_{sk}(g^a)$, $E_{cs}(g^b)$	(7)
Bob $\rightarrow$ *Alice*	:	$E_{sk}(g^b)$	(8)

Protocol 3. C2C-PAKE in a Cross-Realm Setting. KDC_A and KDC_B are key distribution centers which store *Alice*'s password file and *Bob*'s password file. $R(= H_1(g^{xy})), R'(= H_2(g^{x'y'}))$ and $sk(= H_3(g^{ab}))$ are session keys agreed between *Alice* and KDC_A, *Bob* and KDC_B, and *Alice* and *Bob*, respectively. K is a symmetric key shared between KDC_A and KDC_B. $Ticket_B = E_K(g^{pwa \cdot r}, g^r, ID(A), ID(B), L)$. $cs = H_5(g^{pwa \cdot pwb \cdot r \cdot r'})$.

2. KDC_A obtains g^x by decrypting $E_{pwa}(g^x)$, chooses $y, r \in Z_p^*$ randomly and computes $E_{pwa}(g^y)$ and $g^{pwa \cdot r}$. KDC_A also specifies L, a lifetime of $Ticket_B$. Then KDC_A makes $Ticket_B$ and sends $E_R(g^x \oplus g^r, ID(A), ID(B))$, $Ticket_B$ and $E_{pwa}(g^y)$ to *Alice*. Upon receiving the message from KDC_A, *Alice* computes a session key R and decrypts $E_R(g^x \oplus g^r, ID(A), ID(B))$ to find g^r.
3. *Alice* just forwards $Ticket_B$ to *Bob*.
4. *Bob* chooses $x' \in Z_p^*$ randomly and computes $E_{pwb}(g^{x'})$. Then he sends $E_{pwb}(g^{x'})$, ID(A) and ID(B) to KDC_B with $Ticket_B$.
5. KDC_B obtains $g^{pwa \cdot r}$ by decrypting $Ticket_B$, selects $r' \in Z_p^*$ randomly and computes $g^{pwa \cdot r \cdot r'}$. KDC_B also selects another random number $y' \in Z_p^*$, and computes $R'(= H_2(g^{x'y'}))$. Next, KDC_B computes $E_{R'}(g^{pwa \cdot r \cdot r'} \oplus g^{x'}, ID(A), ID(B))$ using R'. KDC_B finally sends $E_{R'}(g^{pwa \cdot r \cdot r'} \oplus g^{x'}, ID(A), ID(B))$, $E_{pwb}(g^{y'})$ and $E_{H_4(g^{pwa \cdot r})}(g^{pwb \cdot r \cdot r'})$ to *Bob*.
6. *Bob* decrypts $E_{pwb}(g^{y'})$ to find $g^{y'}$, and decrypts $E_{R'}(g^{pwa \cdot r \cdot r'} \oplus g^{x'}, ID(A), ID(B))$ using R' to obtain $g^{pwa \cdot r \cdot r'}$ from $g^{pwa \cdot r \cdot r'} \oplus g^{x'}$. He makes $cs(= H_5(g^{pwa \cdot pwb \cdot r \cdot r'}))$. Then *Bob* chooses a random number $a \in Z_p^*$ and computes $E_{cs}(g^a)$. Finally *Bob* sends $E_{cs}(g^a)$ and $E_{H_4(g^{pwa \cdot r})}(g^{pwb \cdot r \cdot r'})$ to *Alice*.
7. *Alice* computes $H_4(g^{pwa \cdot r})$ with her pwa and g^r. And she gets $g^{pwb \cdot r \cdot r'}$ by decrypting $E_{H_4(g^{pwa \cdot r})}(g^{pwb \cdot r \cdot r'})$ with $H_4(g^{pwa \cdot r})$. *Alice* also can compute $cs(= H_5(g^{pwa \cdot pwb \cdot r \cdot r'}))$ using $g^{pwb \cdot r \cdot r'}$ and pwa. Next, *Alice* selects $b \in Z_p^*$ randomly, and computes $sk(= H_3(g^{ab}))$ and $E_{cs}(g^b)$. Finally she sends $E_{sk}(g^a)$ and $E_{cs}(g^b)$ for session key confirmation.

8. After receiving $E_{sk}(g^a)$ and $E_{cs}(g^b)$, *Bob* gets g^b by decrypting $E_{cs}(g^b)$ with cs, and computes $sk(= H_3(g^{ab}))$ with g^b and a. *Bob* verifies g^a by decrypting $E_{sk}(g^a)$ with sk. *Bob* sends $E_{sk}(g^b)$ to *Alice* to confirm the session key. *Alice* also verifies g^b by decrypting $E_{sk}(g^b)$ with sk. Step **7** and **8** are session key exchange and confirmation stages between *Alice* and *Bob*.

Remarks on *cs*. Note that cs can not be directly used as a session key. If an attacker $\mathcal{A}$ obtains pwa and R, she can always get g^r in Step (2). She also can decrypt $E_{H_4(g^{pwa \cdot r})}(g^{pwb \cdot r \cdot r'})$, and get $g^{pwb \cdot r \cdot r'}$ since she knows $g^{pwa \cdot r}$. Finally she can compute $cs(= H_5(g^{pwa \cdot pwb \cdot r \cdot r'}))$ using pwa and $g^{pwb \cdot r \cdot r'}$. This indicates that perfect forward secrecy is not satisfied, if cs is used for a session key instead of sk. Therefore cs should be used to only encrypt g^a or g^b. Then $\mathcal{A}$, given pwa and R, can not compute $sk(= g^{x'y'})$ since computing sk is equivalent to solving the *Diffie-Hellman problem.* If *Alice* wants to re-generate a session key with *Bob*, then $Ticket_B$ can be re-used during its lifetime L.

Security Analysis of C2C-PAKE in a Cross-Realm Setting: We show that C2C-PAKE in a cross-realm setting resists against all the well-known attacks. The adversary $\mathcal{A}$ is denoted as a probabilistic polynomial time machine $\mathcal{A}(1^k, m)$ where 1^k is security parameter and m is the useful information to $\mathcal{A}$. We assume that it is infeasible for $\mathcal{A}$ to solve the discrete logarithm problem and *Diffie-Hellman problem.* Hence $Pr[Forge_{DLP}(k)]$, $Pr[Forge_{DHP}(k)] \leq \varepsilon(k)$ for some negligible function $\varepsilon(k)$.

- ***Perfect forward secrecy in the DPA***: Let $Adv_{\mathcal{A}}^{P/S}(k)$ be the advantage of $\mathcal{A}$ in attacking *perfect forward secrecy in the DPA*. Then we show $Adv_{\mathcal{A}}^{P/S} \leq Pr[Forge_{DHP}(k)]$.
  - ***Case 1*** Assume that an attacker $\mathcal{A}$ knows pwa. Then $\mathcal{A}$ can find g^x and g^y by decrypting $E_{pwa}(g^x)$ and $E_{pwa}(g^y)$. But $\mathcal{A}(1^k, pwa, g^x, g^y)$ still cannot determine $R(= H_1(g^{xy}))$ because the *Diffie-Hellman problem* is infeasible. If R is given to $\mathcal{A}$, then it can get cs in the flow **(6)**. However, $\mathcal{A}(1^k, pwa, g^x, g^y, g^a, g^b, cs)$ can not compute $sk(= H_3(g^{ab}))$ without solving the *Diffie-Hellman problem* of g^a and g^b. Hence the advantage of $\mathcal{A}$ is bounded above by $Pr[Forge_{DHP}(k)]$ for finding session keys. This fact is expected since our session key generating is based on *Diffie-Hellman assumption.*
  - ***Case 2*** An attacker $\mathcal{A}$ with pwb can easily know $g^{x'}$ and $g^{y'}$ by decrypting $E_{pwb}(g^{x'})$ and $E_{pwb}(g^{x'})$. But these values do not help $\mathcal{A}$ to compute sk or R' in old sessions because session key generation is based on the *Diffie-Hellman problem.* $Adv_{\mathcal{A}}^{P/S}(k)$ again is bounded above by $Pr[Forge_{DHP}(k)]$.
  - ***Case 3*** In this case, we allow $\mathcal{A}$ to get both pwa and pwb. However $\mathcal{A}(1^k, pwa, pwb, g^x, g^y, g^{x'}, g^{y'}, g^a, g^b, cs)$ can not obtain sk, R, R' because it is based on the *Diffie-Hellman problem* to get a common session keys.

  By Case 1,2,3 we concludes $Adv_{\mathcal{A}}^{P/S} \leq Pr[Forge_{DHP}(k)]$.

- ***Denning-Sacco attack in the DPA***: Let $Adv_{\mathcal{A}}^{D/S}(k)$ be the advantage of $\mathcal{A}$ in attacking *Denning-Sacco attack in the DPA*. We show $Adv_{\mathcal{A}}^{D/S} \leq \varepsilon(k)$. We classify an adversary $\mathcal{A}$ into two types. One is an *Insider adversary* and the other is an *Outsider adversary.* Insider adversary, who knows *pwa* or *pwb*, is a legal user of the system while Outsider adversary is not. So, Outsider adversary knows neither *pwa* nor *pwb*.
  – ***Case 1: Outsider Adversary*** $\mathcal{A}(1^k, R, sk, R')$***.*** Outsider adversary, $\mathcal{A}$ with session keys R, sk and R' can determine $ID(A)$, $ID(B)$, L, $g^x \oplus g^r$, $g^{pwa \cdot r \cdot r'} \oplus g^{x'}$, g^a, g^b and all conversations in the protocol. But these values do not help $\mathcal{A}$ mount a dictionary attack on *pwb* or *pwa*. For verifying pwa', pwb', $\mathcal{A}$ must get $[g^x, g^y, g^{x'}, g^{y'}]$. But $\mathcal{A}$ can not get x, y, x', y' since these values are ephemeral values of each party. Hence the probability of mounting a dictionary attack in this case is bounded by the probability of finding x, y, x', y' in the Z_p^*. Therefore $Adv_{\mathcal{A}}^{D/S}(k) \leq \frac{1}{|p|}$.
  – ***Case 2: Insider Adversary*** $\mathcal{A}(1^k, pwa, R, sk, R')$***.*** Assume $\mathcal{A}$ is an Insider adversary with *pwa*, and given R, sk and R'. We are going to show that $\mathcal{A}$ can not mount a dictionary attack on *pwb*. First $\mathcal{A}$ can compute $g^{pwa \cdot r \cdot r'} \oplus g^{x'}$ using R', but can not get $g^{pwa \cdot r \cdot r'}$ because $\mathcal{A}$ does not know $g^{x'}$ encrypted with *pwb*. $g^{pwa \cdot r \cdot r'}$ is a critical value to prevent $\mathcal{A}$ from mounting a dictionary attack against *pwb*. If $g^{pwa \cdot r \cdot r'}$ is known to $\mathcal{A}$, then $\mathcal{A}$ can mount a dictionary attack against *pwb*, as follows.
  - Step 1. $\mathcal{A}$ obtains $g^{r \cdot r'}$ using $g^{pwa \cdot r \cdot r'}$ and *pwa*.
  - Step 2. $\mathcal{A}$ gets $g^{pwb \cdot r \cdot r'}$ by decrypting $E_{H(g^{pwa \cdot r})}(g^{pwb \cdot r \cdot r'})$ using *pwa* and g^r.
  - Step 3. $\mathcal{A}$ chooses a candidate password pwb', computes $g^{pwb' \cdot r \cdot r'}$ and compares $g^{pwb' \cdot r \cdot r'}$ with $g^{pwb \cdot r \cdot r'}$ of Step 2. Finally she is able to reduce the size of candidate passwords.
  Therefore $g^{pwa \cdot r \cdot r'}$ must be blinded using $g^{x'}$. In this case, our scheme is strong against a *Denning-Sacco attack in the DPA* with the probability of finding x', y' in $E_{pwb}(g^{x'})$, $E_{pwb}(g^{y'})$. So, $Adv_{\mathcal{A}}^{D/S}(k) \leq \frac{1}{|p|}$.
  – ***Case 3: Insider Adversary*** $\mathcal{A}(1^k, pwb, R, sk, R')$***.*** Assume $\mathcal{A}$ is an Insider adversary with *pwb*. $\mathcal{A}(1^k, pwb, R, sk, R')$ can not mount a dictionary attack against *pwa* since $\mathcal{A}$ can not get g^x encrypted with *pwa*. If $\mathcal{A}$ is given g^x, then $\mathcal{A}$ can mount a dictionary attack against *pwa*, as follows.
  - Step 1. $\mathcal{A}$ obtains g^r from $E_R(g^x \oplus g^r)$ using g^x and R.
  - Step 2. $\mathcal{A}$ can get $g^{pwa \cdot r \cdot r'}$ from $E_{R'}(g^{pwa \cdot r \cdot r'} \oplus g^{x'})$ because $\mathcal{A}$ knows R' and $g^{x'}$.
  - Step 3. $\mathcal{A}$ selects a candidate pwa' and computes $H_4(g^{pwa' \cdot r})$. Then $\mathcal{A}$ decrypts $E_{H_4(g^{pwa \cdot r})}(g^{pwb \cdot r \cdot r'})$ with $H_4(g^{pwa' \cdot r})$, then gets $g^{pwb \cdot r_1 \cdot r_2}$ where r_1 and r_2 are unknown values. She can remove *pwb* from $g^{pwb \cdot r_1 \cdot r_2}$, and get $g^{pwa' \cdot r_1 \cdot r_2}$ using candidate password pwa'. $\mathcal{A}$ compares $g^{pwa' \cdot r_1 \cdot r_2}$ with $g^{pwa \cdot r \cdot r'}$ of Step 2. If two values are matched, the guess is correct. Otherwise $\mathcal{A}$ eliminates a candidate pwa' from her dictionary.

Hence, the hiding g^x does not allow $\mathcal{A}(1^k, pwb, R, sk, R')$ to mount a dictionary attack against pwa. Our scheme is strong against *Denning-Sacco attack in the DPA* with the probability of finding x, y in $E_{pwa}(g^x)$, $E_{pwa}(g^y)$. Hence $Adv_{\mathcal{A}}^{D/S}(k) \leq \frac{1}{|p|}$.

- ***Dictionary attack in the DPA***: Let $Adv_{A}^{D/A}$ be the probability that adversary $\mathcal{A}$ can mount a dictionary attack in the DPA. We analyze this attack in two cases.
  - ***Case 1: Compromise of pwa.*** An attacker $\mathcal{A}$ with pwa can get $g^{pwb\cdot r\cdot r'}$ by decrypting $E_{H_4(g^{pwa\cdot r})}(g^{pwb\cdot r\cdot r'})$ using g^r and its memorable pwa. But $\mathcal{A}$ can not get $g^{pwa\cdot r\cdot r'}$ from $E_{R'}(g^{pwa\cdot r\cdot r'} \oplus g^{x'})$ since $\mathcal{A}$ is not able to get R'. Even if $\mathcal{A}$ gets R', it can not get $g^{x'}$. Hence $\mathcal{A}$ is not able to mount a dictionary attack against pwb, as analyzed in *Denning-Sacco attack in the DPA*. $\mathcal{A}$ can mount a dictionary attack if it solves the *Discrete Logarithm problem* and gets $g^{x'}$. So $Adv_{A}^{D/A}$ is bounded by $Pr[Forge_{DLH}(k)] + \frac{1}{|p|}$.
  - ***Case 2: Compromise of pwb.*** $\mathcal{A}$ with pwb can get $g^{x'}$ and $g^{y'}$. But $\mathcal{A}$ can not mount a dictionary attack against pwa. Even if R' is given to $\mathcal{A}$, it can not mount a dictionary attack as analyzed in *Denning-Sacco attack in the DPA*. Hence, $Adv_{A}^{D/A}$ is negligible.
- ***On-line Guessing attack.***: This attack is detectable, and thwarted by counting the number of failures. If $\mathcal{A}$ is rejected R times, it can reduce the size of the possible set. So, the success probability of on-line guessing attack is bounded by $\frac{1}{|D|-R}$.
- ***Man in the middle attack***: A man-in-the-middle attack requires an attacker $\mathcal{A}$ to fool both sides in a legitimate conversation. $\mathcal{A}$ intercepts $ID(A)$, $ID(B)$ and $E_{pwa}(g^x)$. $\mathcal{A}$ selects a candidate password pwa' and computes $g^{\widetilde{x}}$. Then $\mathcal{A}$ generates $E_{pwa'}(g^{\widetilde{x}})$ and sends it to KDC_A. KDC_A finds $g^{\widehat{x}}$ by decrypting $E_{pwa'}(g^{\widetilde{x}})$ with pwa and computes $\widehat{R}(= H_1(g^{\widehat{x}y}))$. But this value is different from the value $\widetilde{R}(= H_1(g^{\widetilde{x}y}))$ which is able to be computed by $\mathcal{A}$ with pwa'. $\mathcal{A}$ also can not fool both KDC_B and *Bob* without obtaining pwb. The case of attack in the middle of *Alice* and *Bob* are similar to the above case. Consequently $\mathcal{A}$ without obtaining pwa or pwb can not fool any sides in a legitimate conversation.
- ***Replay attack***: The probability of success with regard to a replay attack is trivially negligible because x, y, x' and y' are ephemeral parameters of both parties in a session. Therefore an attacker can not impersonate both parties.

From the analysis above, we conclude that

$$Adv_{\mathcal{A}}^{DPA}(k) < \mathcal{O}(1/(|D| - R)) + Adv_{\mathcal{A}}^{F/S}(k) + Adv_{\mathcal{A}}^{D/S}(k) + Adv_{\mathcal{A}}^{D/A}(k) + \varepsilon(k).$$

On-line guessing attack is unavoidable in any password-authenticated key exchange scheme and its success probability is a lower bound of the success probability of an adversary. Since $Adv_{\mathcal{A}}^{F/S}(k)$, $Adv_{\mathcal{A}}^{D/S}(k)$ and $Adv_{\mathcal{A}}^{D/A}(k)$ have been shown to be negligible, the proposed scheme is secure.

5 C2C-PAKE DPA in a Single-Server Setting

In this section, we propose C2C-PAKE in a single-server setting. Single-server setting, as mentioned in section 2, provides password-authenticated key exchange between two clients on the same server. We further classify the single-server setting into single-server ticket type and single-server non-ticket type. First we introduce a simple C2C-PAKE in the single-server non-ticket type.

C2C-PAKE in the Single-Server Non-ticket Type

The first scheme classified into single-server non-ticket type is 3-Party EKE [13], where each user uses his own password and the third party(KDC) participates in every session during authentication between users. Protocol 4 is a variant of original 3-Party EKE.

As shown in *Protocol 4*, KDC acts as a key transmission center which only stores password information. When *Alice* wants to communicate with *Bob* registered in the same KDC, she sends $E_{pwa}(g^a)$ to KDC. Since KDC knows pwa, it can get g^a by decrypting $E_{pwa}(g^a)$. And KDC selects its own random number $s \in Z_p^*$, computes and sends $E_{pwb}(g^{a \cdot s})$ to *Bob*. *Bob* computes a session key $sk(= H_1(g^{a \cdot b \cdot s}))$. *Bob* also chooses a random number $b \in Z_p^*$ and sends back $E_{pwb}(g^b)$ to KDC. Finally KDC decrypts and sends $E_{pwa}(g^{b \cdot s})$ to *Alice*. *Alice* also computes $sk(= H_1(g^{a \cdot b \cdot s}))$. *Protocol 4* only provides key exchange using passwords, and does not provide session key confirmation. But we can easily design key confirmation stage by adding extra flows to the protocol.

$Alice \rightarrow KDC$	:	ID(A), ID(B), $E_{pwa}(g^a)$	(1)
$KDC \rightarrow Bob$	:	ID(B), $E_{pwb}(g^{a \cdot s})$	(2)
$Bob \rightarrow KDC$	:	ID(A), $E_{pwb}(g^b)$	(3)
$KDC \rightarrow Alice$	:	ID(A), $E_{pwa}(g^{b \cdot s})$	(4)

Protocol 4. C2C-PAKE in the Single-server Non-ticket Type

C2C-PAKE in the Single-Server Ticket Type

C2C-PAKE in the single-server ticket type protocol is easily constructed by modifying C2C-PAKE in a cross-realm setting. C2C-PAKE in a cross-realm setting involves a shared key K so that each *Kerberos* realm can interoperate with *Kerberos* in other realm. We construct a single-server ticket type by converting this shared key(K) between two *Kerberos* into private key(PK) of KDC. The rest part is identical to those of C2C-PAKE in a cross-realm setting.

References

1. M. Bellare, D. Pointcheval and P. Rogaway, “Authenticated key exchange secure against dictionary attacks”, Eurocrypt’00, LNCS Vol.1807, pp.139-155, Springer-Verlag, 2000.
2. S. Bellovin and M. Merrit, “Encrypted key exchange: password based protocols secure against dictionary attacks”, In *Proceedings of the Symposium on Security and Privacy*, pp.72-84, IEEE, 1992.

3. C. Boyd, A. Mathuria, "Key establishment protocols for secure mobile communications: A selective survey", ACISP'98, LNCS Vol.1438, pp.344-355, Springer-Verlag, 1998.
4. V. Boyko, P. MacKenzie, and S. Patel, "Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman", Eurocrypt'00, LNCS Vol.1807, pp.156-171, Springer-Verlag, 2000
5. G. D. Crescenzo, O. Kornievskaia, "Efficient kerberized multicast in a practical distributed setting", ISC'01, LNCS Vol.2200, pp.27-45, Springer-Verlag, 2001.
6. D. Denning, G. Sacco, "Timestamps in key distribution protocols", *Communications of the ACM*, Vol.24, No.8, pp.533-536, 1981.
7. D. Jablon, "Strong password-only authenticated key exchange", *Computer Communication Review*, Vol.26, No.5, pp.5-26, 1996.
8. O. Goldreich and Y. Lindell, "Session-Key Generation Using Human Passwords Only", Crypto'01, LNCS Vol.2139, pp.408-432, Springer-Verlag, 2001.
9. B. Jaspan, "Dual-workfactor encrypted key exchange : Efficiency preventing password chaining attacks", In *Proceedings of the sixth annual USENIX security conference*, pp.43-50, July 1996.
10. M. Hur, B. Tung, T. Ryutov, C. Neuman, A. Medvinsky, G. Tsudik, and B. Sommerfeld, "Pulbic key cryptography for cross-realm authentication in kerberos", *Internet draft*, May 2001.
11. J. Katz, R. Ostrovsky and M. Yung, "Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords", Eurocrypt'01, LNCS Vol.2045, pp.475-494, Springer-Verlag, 2001.
12. S. Lucks, "Open key exchange: How to defeat dictionary attacks without encryting public keys", *The security Protocol Workshop '97*, pp.79-90, 1997.
13. M. Steiner, G. Tsudik, and M. Waider, "Refinement and extension of encrypted key exchange", *ACM Operation Sys. Review*, Vol.29, No.3, pp.22-30, 1995.
14. S. P. Miller, B. C. Neuman, J. I. Schiller, J. H. Saltzer, "Kerberos Authentication and Authorization System", *Section E.2.1, Project Athena Technical Plan*, M.I.T. October 1988.
15. T. Wu, "Secure Remote Password Protocol", In *Proceedings of the Internet Society Network and Distributed System Security Symposium*, pp.97-111, 1998.
16. T. Wu, "A Real-World Analysis of Kerberos Password Security", In *Proceedings of the Internet Society Network and Distributed System Security Symposium*, 1999.
17. V. Varadharajan and Y. Mu, "On the Design of Security Protocols for Mobile Communications", In *Proceedings of Twelfth Annual Computer Security Applications Conference*, pp.78-87. IEEE Computer Society Press, 1996.

Robust, Privacy Protecting and Publicly Verifiable Sealed-Bid Auction

Kun Peng, Colin Boyd, Ed Dawson, and Kapali Viswanathan

Information Security Research Centre
Queensland University of Technology
2, George Street, Brisbane, QLD 4001, Australia
{k.peng,c.boyd,e.dawson,k.viswanathan}@qut.edu.au

Abstract. The sealed bid model of auctions is ideally suited for electronic auction systems, as they avoid the requirement for real-time communications between various entities. A sealed bid auction scheme designed by Kikuchi, Harkavy and Tygar is analysed. Several shortcomings are identified in this scheme including the lack of public verifiability and the possibility of collusion between bidders and/or auctioneers. A new scheme is designed to overcome these shortcomings.

1 Introduction

Sealed-bid auction systems require mechanisms for providing confidentiality and integrity services to the bid values until a bid-opening phase occurs. In order to electronically simulate the sealing process, suitable cryptographic mechanisms are employed. Electronic sealed-bid auction systems tend to be more resilient to real-time synchronisation problems than do electronic outcry auctions.

Since the proposal by Franklin and Reiter [5] many other sealed bid e-auction systems have been presented. Sakurai and Miyazaki [16] proposed a sealed-bid, Dutch-style[1] auction system using undeniable signatures. Although this proposal possesses good security properties, it is both computationally and communicationally expensive. Sako [15] proposed an alteration to reduce the communication overhead by requiring the bidders to encrypt the bid value for the auctioneers, rather than use the undeniable signature technique. Although the communication cost was greatly reduced, the most interesting properties of the proposal by Sakurai and Miyazaki, namely user-controlled confidentiality for the sealed bid and public verifiability, were neglected. Suzuki *et al.* [19] employed hash chains along with a standard signature scheme to greatly improve the computational efficiency for the proposals by Sakurai and Miyazaki, and Sako. The number of rounds of communication between the bidders and the auctioneer remains high, as was the case with the proposal by Sakurai and Miyazaki.

The aim of this paper is to improve the security properties of the proposal by Kikuchi, Harkavy and Tygar [9], which employs the homomorphic property of Shamir's secret sharing scheme [17]. Henceforth, the proposal by Kikuchi *et al.* will be called the KHT

[1] In Dutch style auction systems, to determine the maximum price for the goods the auctioneer starts with the highest value and progressively reduces the bid value. This process stops when the first bidder is willing to pay the bid value.

system. Although this is an efficient proposal, it is an incomplete auction system because crucial security properties, such as non-repudiation, were not discussed. The most notable absence is public verifiability of the auction process. Additionally, the method to identify the winning bidder is unreliable and the scheme is vulnerable to certain attacks, discussed below. However, the use of secret sharing techniques and suitable homomorphisms has the potential to achieve most of the security properties required by secure e-auction systems in the most efficient manner. The proposal in this paper provides public verifiability, achieves a more robust mechanism to identify the winning bidder and prevents the attacks identified in the KHT system. The scheme requires only a single, non-interactive communication between the auctioneers and the bidders.

Section 2 presents the desired properties of sealed-bid e-auction system followed by Section 3, which presents a brief description and analysis of the KHT system. Section 4 introduces necessary cryptographic protocol constructs employed in the proposal. Section 5 presents the proposal for a new auction scheme followed by the security analysis in Section 6 and a comparison with the KHT scheme in Section 7.

2 Properties of Sealed-Bid e-Auction Systems

Sealed-bid auction systems consist of at least three phases, which are:

Bid submission: all the bidders will be expected to seal their bid value and submit the sealed-bid to the auctioneer;

Bid opening: the auctioneers, with or without the assistance of the bidders, appropriately open the sealed-bids; and,

Winner identification: the auctioneers identify the winning bid and the winning bidder according to the established auction rules.

The following properties that are desirable in secure electronic auction systems have been identified in the literature [6,10,18]. Although an extensive list is available, it appears that not all properties can be achieved by any one e-auction system.

1. Correctness: If every party acts honestly, the correct winning price and winner(s) will be identified according to the auction rules.
2. Fairness includes:
   - No bidder knows anything about other bids before he submits his own bid. This is also included in confidentiality.
   - After a bidder submits his bid, the bid cannot be modified.
   - No bidder can deny his bid after he submits it. This is sometimes called non-repudiation of bids.
3. Robustness: Malicious behaviour of any party should not compromise the system or lead to an incorrect result. Robustness is complementary to correctness and guarantees that if there is a result, that result must be correct no matter what system failure or attack may occur. Especially the following three attacks must be prevented.
   (a) One or more bidders conspire with the auctioneer(s) to subvert the auction rules. We will call this the *ABC (auctioneer-bidder collusion) problem.*
   (b) Some bidders collude to subvert the auction rules. This is called the *BBC (bidder-bidder collusion) problem.*

   (c) A subset of the auctioneers declare that a bidder has not submitted any bid or has submitted an incorrect bid, while the bidder claims to have submitted a correct bid. The bidder cannot prove his innocence without revealing the bid. This situation is said to be a *dispute*.
4. Confidentiality of sealed-bid: The auctioneers must not be able to determine the value of the sealed-bid until the bid opening phase.
5. Anonymity: The identities of the losing bidders remain confidential after the winner identification phase.
6. Privacy of losing bids: The losing bids remain confidential, even for the auctioneer, after the winner identification phase.
7. Public Verifiability: All the participants in the auction system, as well as a neutral observer, must be able to verify the validity of the critical operations performed by the auctioneers. Critical operations are those that have the capability to alter the correct result of the auction.

3 Auction Scheme Proposed by Kikuchi, Harkavy and Tygar

3.1 Description of KHT

Like some other auction schemes, the KHT scheme employs the homomorphic property of Shamir's secret sharing to efficiently calculate the winning price and hide the losing bids. Another paper by Kikuchi *et al.* [7] employs a very similar technique. A threshold number of auctioneers cooperate to open the sealed-bids. Their scheme presupposes an established set of biddable prices. A brief description of their proposal is as follows:

System setup. For each bidder, the auctioneers create a special identification tag, which is a signature on the identity of the bidder, I, using the auctioneers' secret signing key. The identification tags are shared secrets between the auctioneers and the corresponding bidder. For each biddable price, the bidder calculates the identification token, TK_i, by encrypting the identification tag along with a random value.

Bid submission. Every bidder generates a polynomial for each biddable price. The constant coefficients of the polynomial is TK_i if the bidder is willing to pay the price, or zero otherwise. The other coefficients of the polynomials are independently and randomly chosen by the bidder. Every bid is shared among the auctioneers.

Determination of the winning bid. Every auctioneer adds the shares for a biddable price, sent by different bidders, to form a *general share*. Then the auctioneers pool the general shares to calculate the sum of all bids at that price using Lagrange interpolation. The sum will be zero if no bidder is willing to pay that price, or the sum of some identification tokens. A binary search is used to identify the winning price.

Identification of the winning bidder. The auctioneers determine the winner by decrypting the identification token to obtain the identification tag. The identification tag of the winner and the corresponding random value are published for verification.

Due to the homomorphic property of the secret sharing algorithm, the computational cost to open the sealed-bids is independent of the number of bidders. The binary search strategy can be employed to find the winning price, thereby resulting in a more efficient system. The losing bids are kept secret, unless the number of colluding auctioneers exceeds the threshold of the secret sharing algorithm.

3.2 Shortcomings

The following shortcomings can be identified in the KHT system:

Vulnerability of the winner identification technique The method to identify the winner is not reliable. A problem is the tie situation, when the auctioneers can only find the winning price but cannot determine the identities of winning bidders.
Another problem is that the scheme is vulnerable to malicious bidders, who may submit illegally formatted identification tokens. Such a vulnerability exists because the auctioneers cannot *verify* the format of the identification tokens submitted by individual bidders without reconstructing all the submitted bids. Reconstructing all the submitted bids is essential to identify the illegal bids, but it will violate the privacy of the bid values of honest bidders.

***ABC* attack:** A bidder may conspire with one or more auctioneers to change his bid. This is possible because there is no public commitment for the bid in their scheme, since the bid is a shared secret between the bidder and the auctioneers. So if a bidder conspires with at least one auctioneer, the bidder can illegally change the bid to gain advantage over other bidders.

***BBC* attack:** The use of homomorphic secret sharing mechanisms may result in the *BBC* problem. If there is no collusion among the bidders and a general share is equal to zero, all the shares to form it are zero with a very high probability. If some bidders collude they can exploit the homomorphic property to make the sum of bids at a price to be zero while some bids at that price are non-zero. For example, for a particular price, the bids of two bidders can be k and $-k$. Their bids indicate that they are willing to pay the price, while the sum of their bids indicates that neither of them are willing to pay the price.

Dispute: An auctioneer declares that a bidder failed to submit a share or the submitted share from a bidder is invalid while the bidder insists that the share is valid. This *dispute* situation cannot be solved within the framework of the KHT system.

Due to these problems, the KHT system is not a practical system. Although there are other schemes [8,7,4,1] that employ similar techniques used in the KHT system, these schemes do not address all the problems identified in this section.

4 Verifiable Secret Sharing

The proposed scheme requires a verifiable secret sharing (VSS) scheme that can share binary valued secrets. Namely the secret could be zero or non-zero. General purpose VSS schemes, such as the proposal by Benaloh [3] and Pedersen [13], do not possess this property and, therefore, cannot be employed in our proposal. This is because such schemes are not information-theoretically hiding. So, the information-theoretically secure scheme proposed by Pedersen [14] is employed.

The VSS scheme can be employed in conjunction with an appropriate verifiable encryption scheme to realise a publicly verifiable secret sharing (PVSS) mechanism. The verifiable encyption primitive can be implemented by modifying the verifiable encyption scheme by Bao [2]. The only difference is that in Bao's scheme only one value is proved to be encrypted correctly whereas for Perdersen's scheme two values must be encrypted. The encryption scheme can be the algorithm by Okamoto and Uchiyama [11] as suggested by Bao or other related encryption schemes like Paillier's [12].

5 The Auction Scheme

We describe a new auction scheme, which includes several improvements to the KHT system. For each biddable price, the willingness to pay is denoted by a random non-zero value and unwillingness is denoted by the zero value. The bidders act as the dealers of a secret sharing scheme and the auctioneers act as the share-holders. The bidders share the bids corresponding to different biddable prices using the VSS scheme as described in Section 4. The shares are communicated to the auctioneers using a secure channel. The commitments generated by the VSS scheme are published in a bulletin board, which has read-only access for the public and write-access for a trusted bulletin-board manager. *All communications by the bidder to the auctioneers must be through the bulletin-board.* To verify the validity of their shares, the auctioneers can use the commitment information published in the bulletin board by the bidders.

A threshold number of auctioneers are trusted not to recover all the bids by every bidder. Instead, they only recover the sum of all the bids sent to them by various bidders for a *particular price*. This sum of bids will be called the *general share*. If the general share is zero then, with a high probability, none of the bidders are willing to pay the price and the auctioneers can proceed to recovering the general share for the next price. If the general share is non-zero, then the auctioneers identify the winning bidder by recovering the individual bids sent by the bidders for that *particular price*, which is the winning price. This process protects the privacy of the losing bids. In order to protect the identities of the bidders, pseudonyms are used during bidding which can be correlated with the bidder only by the bidder and a trusted certifying entity.

In the case of a dispute about the identification of the winning bidder or bid, all the bidders are required to encrypt the shares corresponding to various auctioneers and bids using the verifiable encryption technique discussed in Section 4. This converts the private verification procedure for the VSS commitments into a public verification procedure. Such an *optimistic* use of public verification greatly reduces the computational load on the bidders when there is no dispute. Moreover, it protects the value of the bids and the identity of the bidders when there is a dispute.

Figure 1 illustrates the procedure for the auction when all participants act honestly. The subsequent sections will provide the detailed description of the auction.

5.1 Parameter Initialization

There are three kinds of participants in the scheme: n bidders b_i for $i \in Z_n, m$ auctioneers for $j \in Z_m$ and an oblivious third party T. The auctioneers are trusted not to recover the bids of the losing bidders. T is trusted only for providing a trust-based anonymity service for the bidders.

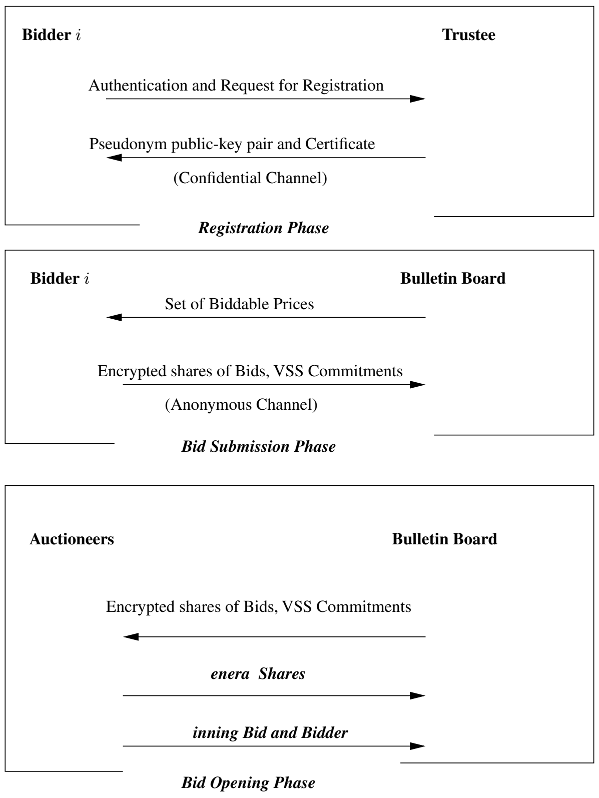


Fig. 1. Auction Procedure: The Optimistic Case

- There is a bulletin board, where public information can be published by all the parties, and cannot be modified or removed.
- There are w biddable prices p_l for $l \in Z_w$. They are published by the auctioneers on the bulletin board.
- The auctioneers choose large primes p and q such that q divides $p-1$. G_q of order q is a subgroup of Z_p^*.
- g and h are independently chosen primitive elements of G_q so that nobody knows $\log_g h$. They are published by the auctioneers on the bulletin board.
- Every auctioneer publishes his public key certificate on the bulletin board. Encryption for the j^{th}auctioneer is denoted by $E_j()$ with corresponding decryption function $D_j()$. We will assume that this encryption can be extended for public verification when required. In practice a separate encryption algorithm could be used for this purpose.
- Every bidder b_i has a long-term certified public key y_i and corresponding private key x_i.

$E(x, y) = g^x h^y$ denotes the commitment information for x. The sharing threshold is k, which means k or more correct shares are enough to reconstruct the secret.

5.2 Registration Phase

Every bidder authenticates, in a suitable fashion to T. The bidder b_i and T perform the following exchanges in a secure channel.

1. T securely chooses a private key $\hat{x}_i$ and computes the corresponding public key $\hat{y}_i$, which will be the pseudonym for b_i;
2. T commits to the private key and the pseudonym, and signs the public key and the commitment as follows: $Com\_T_i = H(b_i, r_i)$ and $Sig\_T_i = SIG_T(\hat{y}_i, Com\_T_i)$, where r_i is a random integer chosen by T
3. T sends to b_i the private key $\hat{x}_i$, the random integer r_i and the certificate $C_i = (\hat{y}_i, Com\_T_i, Sig\_T_i)$.
4. b_i verifies $Com\_T_i = H(b_i\ , r_i)$ and $Sig\_T_i = SIG_T(\hat{y}_i, Com\_T_i)$.

5.3 Bid Submission Phase

Every bidder, b_i, determines his/her evaluation $\hat{l} \in Z_w$, sets $\{s_{i,l} = 0 \mid l > \hat{l}\}$ and $\{s_{i,l} = R_{i,l} \in_R Z_q$ and $R_{i,l}\ /= 0| \ l \le \hat{l}\}$. Every bidder chooses w random integers $t_{i,l} < q$ and $l \in Z_w$ and performs the following steps to submit the bids.

Calculation of shares: The secret bids for each biddable price are shared for each auctioneer using the following equations:

$$s_{i,l,j} = f_{i,l}(j+1)$$
$$t_{i,l,j} = g_{i,l}(j+1)$$

for all $l \in Z_w, j \in Z_m$ and where:

- $f_{i,l,r}$ and $g_{i,l,r}$ are secret random values chosen from Z_q by the bidder for $r = 1, 2 \ldots k-1$.
- $f_{i,l}(x) = \sum_{r=0}^{k-1} f_{i,l,r}x^r \bmod q$, $f_{i,l,0} = s_{i,l}$, $g_{i,l}(x) = \sum_{r=0}^{k-1} g_{i,l,r}x^r \bmod q$ and $g_{i,l,0} = t_{i,l}$

Calculation of public commitments: The commitments for the bids $s_{i,l}$ for each price and the commitments for the coefficients of the polynomial for each price are calculated using the following equations:

$$\{E_{i,l,0} = E(s_{i,l}, t_{i,l}) \mid \forall l \in Z_w\}$$
$$\{E_{i,l,r} = E(f_{i,l,r}, g_{i,l,r}) \mid \forall l \in Z_w, \forall r \in Z_k \setminus \{0\}\}$$

Publication of Commitments: The public information PUB_i, which includes the following information, are published in the bulletin board:

- the certificate C_i;
- the set of commitments, $\{E_{i,l,r} \mid \forall l \in Z_w, \forall r \in Z_k\}$; and,
- the signature using the pseudonym on the set of bids, $\hat{\sigma}_i = SIG_{\hat{x}_i}(\{\hat{y}_i || E_{i,l,r} \mid \forall l \in Z_w, \forall r \in Z_k\})$
- the signature $\sigma_i = SIG_{x_i}(\hat{x}_i || \{E_{i,l,r} \mid \forall l \in Z_w, \forall r \in Z_k\})$

The last term in the published information, σ_i, cannot be publicly verified in this stage even though the public key corresponding to the private key x_i is publicly available. This is because the message that corresponds to this signature is not public, as $\hat{x}_i$ is a shared secret between the bidder and the trustee, T (refer to the registration phase). This property is essential to protect the anonymity of the bidders. This signature information is important for the bidder to be certain that the trusted party, T, did not submit the bid. T is trusted only to protect the identity of the losing bid. So this signature is very useful to limit trust on T.

Communication of shares for the auctioneers: The shares for the j^{th} auctioneer for every secret bid are collated to form $B_{i,j} = (\hat{y}_i, \{s_{i,l,j}, t_{i,l,j} \mid \forall l \in Z_w\})$ and the bidder sends the following information to the bulletin board:

$$V_{i,j} = E_j(B_{i,j}, SIG_{\hat{x}_i}(B_{i,j}))$$

5.4 Bid Opening Phase

If non-repudiation of the winning bid is essential, then T may verify σ_i in the data structure PUB_i, for every bidder b_i and every sealed bid available in the bulletin board. This process will prevent the winning bidder from repudiating the bid. At the end of the bid submission phase, the auctioneers open the bids as follows.

Signature Verification. For each sealed bid, PUB_i, available in the bulletin board:

– The auctioneers verify the certificate, C_i and the signature, $\hat{\sigma}_i$, in PUB_i using $\hat{y}_i$.
– The auctioneer j decrypts $V_{i,j}$ and verifies the signature on $B_{i,j}$ using $\hat{y}_i$.

If either of the verifications fail for any anonymous bidder, $\hat{y}_i$, then the bidder and his bid are removed from the auction proceedings and the value $B_{i,j}$ is published on the bulletin board for public verification of the failed signature tuples.

Recovering the Winning Price. We assume that the winning price is the highest bid. A binary search strategy can be employed to determine the winning price. The following steps are iterated in a suitable fashion until a possible winning price $\hat{p}_W$ is found:

1. Summing up the Bids
Every auctioneer d_j adds up all the bids for price p_l sent to him to form his general share for that price.

$$\{\hat{s}_{l,j} = \sum_i s_{i,l,j}, \hat{t}_{l,j} = \sum_i t_{i,l,j} \mid \forall j \in Z_m, \forall i \in Z_n\}$$

These general shares are published on the bulletin board and their correctness can be verified by anyone by the following equation:

$$\{E(\hat{s}_{l,j}, \hat{t}_{l,j}) = \prod_{r=0}^{k-1}(\prod_{i=0}^{n-1} E_{i,l,r})^{(j+1)^r} \mid \forall j \in Z_m\} \quad (1)$$

If the above equation is satisfied for no less than k general shares, go to next step. Otherwise a random invalid general share $\hat{s}_{l,j}$ is chosen to be corrected. In this case, d_j must point out an invalid original share $s_{i,l,j}$, otherwise the auctioneer is blacklisted for incorrectly summing up the original shares. The auctioneer may provide an unsuccessful verification of the following equation to accuse some bidder, $\hat{y}_i$, of submitting incorrect share $(s_{i,l,j}, t_{i,l,j})$.

$$E(s_{i,l,j}, t_{i,l,j}) = \prod_{r=0}^{k-1} E_{i,l,r}^{(j+1)^r} \tag{2}$$

If $\hat{y}_i$ denies the accusation then a *dispute* said to have occured. To solve the dispute, $\hat{y}_i$ is required to publish verifiable encryptions of the shares $s_{i,l,j}$ for each auctioneer. If they are valid, then valid original shares $s_{i,l,j}$ can be decrypted from them and $\hat{s}_{l,j}$ is recalculated. Otherwise the auctioneers ask T to recover the true identity, b_i of $\hat{y}_i$. In such a case, all shares and public information from $\hat{y}_i$ are removed from the auction proceedings. This step is iterated until there are enough correct general shares.

2. Interpolation
No less than k correct general shares are put together to recover the sum of all the bids, s_l from the set of shares $\{s_{l,j}\}$, at price p_l using Lagrange interpolation. Any outsider can also perform the interpolations to check the validity of winning price.

Identifying the Winner(s).

1. Opening the bids at $\hat{p}_W$
Every bidder's shares at $\hat{p}_W$ are published by the auctioneer. If a bidder disputes his shares, he can correct them. The following verification is performed.
$E(s_{i,W,j}, t_{i,W,j}) = \prod_{r=0}^{k-1} E_{i,W,r}^{(j+1)^r}$ for $i = 0, 1, \ldots, n-1$ and $j = 0, 1, \ldots, m-1$. The verification can be performed by any entity. If no less than k shares pass the verification, his bid at $\hat{p}_W$ is recovered. Otherwise his bids are removed. $\hat{y}_i$ is a possible winner if $s_{i,W} \neq 0$.
2. Upward opening
$s_{i,W+1}$ for i satisfying $s_{i,W} \neq 0$ are opened. If any opened $s_{i,W+1}$ is non-zero, $W \leftarrow W+1$ and $s_{i,W+1}$ for i satisfying $s_{i,W} \neq 0$ are opened. This step is iterated until $s_{i,W+1} = 0$ for every i satisfying $s_{i,W} \neq 0$. This price p_W is the winning price. $\hat{y}_i$ for each i satisfying $s_{i,W} \neq 0$ are the pseudonyms of the winners.
3. Identification
All the winners are required to reveal their real identities by publishing b_i, $\hat{x}_i$ and r_i where the winner is denoted as b_i. If any anonymous winner $\hat{y}_i$ refuses or fails to open his identity the auctioneers can ask T to recover his identity. After verifying that the bidder is actually a winner according to the information available on the bulletin board, T recovers the real identity b_i, and the pseudonym private key $\hat{x}_i$, the random integer, r_i of $\hat{y}_i$. T publishes the so recovered information on the bulletin board. Any entity can verify that $Com\_T_i \overset{?}{=} H(b_i, r_i)$, $Sig\_T_i \overset{?}{=} SIG_T(\hat{y}_i, Com\_T_i)$ and $\sigma_i \overset{?}{=} SIG_{y_i}(\hat{x}_i || \{E_{i,l,r} \mid \forall l \in Z_w, \forall r \in Z_k\})$. If either of the first two verifications fails, T is dishonest and can be held accountable. If the last verification fails, the winning bid is removed and recalculated.

6 Security Analysis

In this section it is illustrated that the properties described in Section 2 are satisfied.

Correctness: If the shares at p_l are correct, namely $E(s_{i,l,j}, t_{i,l,j}) = \prod_{r=0}^{k-1} E_{i,l,r}^{(j+1)^r}$ for $i = 0, 1 \ldots, n-1$ and $j = a_0, a_1 \ldots, a_{k-1}$ where $a_\gamma \in Z_m$ for $\gamma = 0, 1 \ldots, k-1$, then

$$E(\hat{s}_{l,j}, \hat{t}_{l,j}) = g^{\hat{s}_{l,j}} h^{\hat{t}_{l,j}} = g^{\sum_{i=0}^{n-1} s_{i,l,j}} h^{\sum_{i=0}^{n-1} t_{i,l,j}}$$

$$= \prod_{i=0}^{n-1} g^{s_{i,l,j}} h^{t_{i,l,j}} = \prod_{i=0}^{n-1} \prod_{r=0}^{k-1} E_{i,l,r}^{(j+1)^r} = \prod_{r=0}^{k-1} (\prod_{i=0}^{n-1} E_{i,l,r})^{(j+1)^r} = \prod_{r=0}^{k-1} E_{l,r}^{(j+1)^r}$$

and $E_{l,r} = \prod_{i=0}^{n-1} E_{i,l,r}$ for $j = a_0, a_1 \ldots, a_{k-1}$ where $a_\gamma \in Z_m$ for $\gamma = 0, 1 \ldots, k-1$. Thus, the verification equations are correct.

Robustness: The proposed scheme is robust and avoids all the problems described in Section 2. It can ensure that any incorrect result will be noticed by any neutral observers. If the number of honest auctioneers is more than the threshold number, a correct result will definitely be recovered. In our scheme robustness is guaranteed by the following four operations: verification of general shares at the prices on the route of binary search, opening of all the bids at $\hat{p}_W$, the upward opening of the bids of possible winners and the verification of the winner's signature on his bid. The problems identified in the KHT system has been avoided due to the following mechanisms:

1. General share verification guarantees a unique value is computationally bound in the commitment information.
   As demonstrated by Pedersen [14], at a price p_l, satisfaction of (1), namely
   $$E(\hat{s}_{l,j}, \hat{t}_{l,j}) = \prod_{r=0}^{k-1} E_{l,r}^{(j+1)^r} \text{ for } j = a_0, a_1 \ldots, a_{k-1}$$
   where $a_\gamma \in Z_m$ for $\gamma = 0, 1 \ldots, k-1$, guarantees that a unique value $\hat{s}_l$ satisfying $E(\hat{s}_l, \hat{t}_l) = E_{l,0}$ for some $\hat{t}_l$ can be recovered from $\hat{s}_{l,j}$ for $j = a_0, a_1 \ldots, a_{k-1}$ where $a_\gamma \in Z_m$ for $\gamma = 0, 1 \ldots, k-1$ by interpolation.
2. The value $\hat{s}_l$ bound in $E_{l,0}$ is the sum of bids at p_l.
   If we see $E_{i,l,0}$ as a sealed bid and a value $S_{i,l}$ satisfying $E(S_{i,l}, T_{i,l}) = E_{i,l,0}$ as b_i's unchangeable bid at price p_l, $\hat{s}_l$ is always the sum of bids at p_l since
   $$E(\hat{s}_l, \hat{t}_l) = E_{l,0} = \prod_{i=0}^{n-1} E_{i,l,0} = \prod_{i=0}^{n-1} E(S_{i,l}, T_{i,l}) = E(\sum_{i=0}^{n-1} S_{i,l}, \sum_{i=0}^{n-1} T_{i,l})$$
   leads to $\hat{s}_l = \sum_{i=0}^{n-1} S_{i,l}$.
   So $\hat{s}_l$ is always the sum of bids at p_l whether the shares are correct or not. Therefore the true winning price is not less than $\hat{p}_W$.

3. Opening all the bids at $\hat{p}_W$ can identify all the possible winning bids. Compared to the token-based method in KHT proposal, our method is direct and robust.
4. The upward opening of the bids of possible winners guarantees that correct winner(s) are found. After the upward opening,
   – every bidder has at least a zero bid at a price no more than p_{W+1};
   – every loser has at least a zero bid at a price no more than p_W;
   – for every winner b_i, $S_{i,W}\ /=.0$
5. The signature verification $\sigma_i \overset{?}{=} SIG_{y_i}(\hat{x}_i || \{E_{i,l,r} \mid \forall l \in Z_w, \forall r \in Z_k\})$ robustly identifies the winning bidder corresponding to the winning bid. Although T can recover $\hat{x}_i$, only b_i can form valid σ_i because only b_i is assumed to know the private key, x_i, corresponding to the long-term certified public key, y_i.
6. PVSS technique based on verifiable encryption enables a bidder to prove the validity of his bids without revealing them.

Upward opening prevents *BBC problem*. In all the operations the shares are verified against the published commitments before the bids are reconstructed. Thus *ABC problem* is avoided. When there is a dispute between a bidder and the auctioneer, an innocent bidder can employ the PVSS protocol discussed in Section 4 to prove the shares he distributes are correctly committed.

However there is another problem. The bids from a bidder may be inconsistent. If a winner submits a zero bid at a price lower than p_W or a loser submits a non-zero bid at a price higher than p_W some problem may arise. The auctioneers interpret them in one way and obtain a reasonable result. But after the winning price and winners are declared the bidder may publish some of his bids, which can be verified against the corresponding commitment as valid and leads to a reasonable result. The solution is to ignore these protests as long as the auctioneers follow the correct procedure to compute the result. The result recovered by the auctioneers is indisputable because it is uniquely determined by the bids and the pre-established auction rules.

Confidentiality: is achieved because the commitments published before the shares submission reveal no information about the bids, while the encryption scheme is computationally hiding.

Public Verification: Anyone can verify that the auctioneers follow the unique route of binary search, the sum of bids are opened correctly along the search route, every losers has a bid opened correctly at a price no more than p_W as zero, every winner has a bid opened correctly at p_W as non-zero, all winners' bids are opened correctly at p_{W+1} as zero and all winners' commitments are correctly signed by himself. So any protest by any bidder can be judged by any outside observers. Especially PVSS technique enables any neutral party to solve the *dispute*.

Privacy: Under this scheme the privacy of the losing bids can be protected unless more than k auctioneers are malicious and collude to recover them. Even though enough malicious auctioneers collude, they can only know the distribution of the bids. Because all the bids are submitted anonymously, without collusion of T the auctioneers cannot link the losing bids to their owners. Without colluding with a number of auctioneers over the threshold T do not know the bids of the losers. There is an exception. Upward opening may reveal some losing bids. But these bids are submitted by dishonest bidders with a overwhelmingly high probability, it is not necessary to protect their privacy.

7 Comparison with KHT Scheme

Table 1 lists the properties satisfied in by the KHT system and our proposal.

Table 1. Comparison of properties

	Fairness	Confidentiality	Public Verifiability	Privacy of losing bids	Robustness
KHT System	no	conditional	no	conditional	no
Proposal	yes	yes	yes	conditional	yes

The efficiency analysis assumes that in the KHT system RSA signature and ElGamal encryption are used for setting up confidential and authenticated channels between bidders and auctioneers. The following observations can be made on our proposal.

1. Communication Efficiency
   Only one round of communication is needed after the bidder submits their bids.
2. Computation Efficiency
   - The computation load of T is n exponentiations.
   - The computation load of a bidder is $2wk + 3m + 2$ exponentiations. All can be performed off-line.
   - The computation load of the auctioneers is $(k + 2)m(n + \log_2 w) + n(3m + 1)$ exponentiations. In comparison the KHT scheme uses only $3nm + (k + 2)m \log_2 w$ exponentiations.

The proposed scheme achieves more properties than the KHT system but, as mentioned above, has higher computaitonal requirements. For the sake of this analysis, we assume that in Kikuchi's scheme the secret sharing by Pedersen [14] is employed and the general shares at the prices on the route of binary search are verified by the auctioneers. In a typical example with $n = 10$, $w = 1024$, $m = 4$, $k = 2$, the number of exponentiations in KHT system and the proposed scheme are 280 and 450 respectively.

8 Conclusion

By modifying the KHT system, a sealed bid auction system has been designed that provides confidentiality, anonymity for the losing bids and robustness. Although the proposed scheme is more costly than the KHT system, extra security functionalities are accomplished. Future research will attempt to reduce the cost by employing suitable optimisation techniques for various verification procedures. In both the KHT system and the proposal, the winning bid is the highest price value. It would be interesting to extend the technique to determine the second highest bid without identifying the second highest bidder, which may be required in Vickery auction.

References

1. Masayuki Abe and Koutarou Suzuki. M+1-st price auction using homomorphic encryption. In *PKC 2002*, pages 115–124, 2002.
2. Feng Bao. An efficient verifiable encryption scheme for encryption of discrete logarithms. In *CARDIS'98, 1820 of Lecture Notes in Computer Science*, pages 213–220. Springer Verlag, 1998.
3. J C Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Advances in Cryptology (Proceedings of CRYPTO '86. Santa Barbara, CA. August 1986. ed. by A. M. Odlyzko.)*, pages 251–260. Springer Verlag, 1986.
4. Koji Chida, Kunio Kobayashi, and Hikaru Morita. Efficient sealed-bid auctions for massive numbers of bidders with lump comparison. In *LNCS 2200*, page 48, 2001.
5. Matthew K Franklin and Michael K Reiter. The design and implementation of a secure auction service. *IEEE Transactions on Software Enginerring*, 22(5):302–312, May 1996.
6. Michael Harkavy, J D Tygar, and Hiroaki Kikuchi. Electronic auctions with private bids. In *3rd Usenix Workshop on Electronic Commerce*, pages 61–83, 1998.
7. Hiroaki Kikuchi, Shinji Hotta, Kensuke Abe, and Shohachiro Nakanishi. Distributed auction servers resolving winner and winning bid without revealing privacy of bids. In *proc. of International Workshop on Next Generation Internet (NGITA2000), IEEE*, pages 307–312, July 2000.
8. Hiroaki Kikuchi. (m+1)st-price auction. In *Proc. of The Fifth International Conference on Financial Cryptography '01, IFCA*, pages 291–298, February 2001.
9. H Kikuchi, Michael Harkavy, and J D Tygar. Multi-round anonymous auction. In *Proceedings of the First IEEE Workshop on Dependable and Real-Time E-Commerce Systems*, pages 62–69, June 1998.
10. E Magkos, M Burmester, and V Chrissikopoulos. Sam: A flexible and secure auction architecture using trusted hardware. 2000. Available at http://thalis.cs.unipi.gr/∼emagos/EC_WEB2000_Springer.ps.
11. T Okamoto and S Uchiyama. A new public-key encyptosystem as secure as factoring. In *volume 1403 0f Lecture Notes in Computer Science*, pages 308–318. Advances in Cryptology - CRYPTO'98, Springer Verlag, 1998.
12. P Paillier. Public key cryptosystem based on composite degree residuosity classes. In *LNCS 1592*, pages 223–238. Eurocrypt'99, Springer Verlag, 1999.
13. Torben P. Pedersen. Distributed provers with applications to undeniable signatures. pages 221–242. EUROCRYPT 91, Springer Verlag, 1991.
14. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *volume 547 of Lecture Notes in Computer Science*, pages 129–140. Advances in Cryptology - EUROCRYPT 91, Springer Verlag, 1991.
15. K Sako. An auction scheme which hides the bids of losers. In *Public Key Cryptology, PKC'2000, Springer*, pages 422–432, 2000.
16. Kouichi Sakurai and S Miyazaki. A bulletin-board based digital auction scheme with bidding down strategy -towards anonymous electronic bidding without anonymous channels nor trusted centers. In *Proc. International Workshop on Cryptographic Techniques and E-Commerce*, pages 180–187. City University of Hong Kong Press, 1999.
17. Adi Shamir. How to share a secret. *Communication of the ACM*, 22(11):612–613, Nov 1979.
18. Dawn Xiaodong Song and Jonathan K Millen. Secure auctions in a publish/subscribe system. Available at http://www2.csl.sri.com/∼millen/papers/dcca8.ps.
19. Koutarou Suzuki, Kunio Kobayashi, and Hikaru Morita. Efficient sealed-bid auction using hash chain. In *ICISC 2000, LNCS 2015*, pages 183–191, 2000.

Attacking Predictable IPsec ESP Initialization Vectors

Sami Vaarala[1], Antti Nuopponen[1], and Teemupekka Virtanen[2]

[1] Netseal {sami.vaarala,antti.nuopponen}@iki.fi
[2] Helsinki University of Technology tpv@tml.hut.fi

Abstract. Predictable initialization vectors in IPsec ESP encryption, allowed by the IPsec specifications and used by most implementations, compromise IPsec confidentiality. By using an adaptive chosen plaintext attack, an attacker can break low entropy plaintext blocks using brute force, and confirm guesses of the contents of arbitrary plaintext blocks. We analyze the preconditions and the seriousness of such attacks, and provide results of practical attack experiments.

1 Introduction

The IP Security Architecture (IPsec) [6] is widely used for end-to-end connection encryption, for remote access to a protected intranet, and for interconnecting sites using encrypted VPN tunnels. The currently specified IPsec ESP encryption algorithms use cipher block chaining (CBC) mode [9,10]. The initialization vector (IV) is included in the ciphertext of every packet to allow the receiver to decrypt individual packets regardless of packet loss or reordering of packets.

The specifications for ESP DES [9] and other ciphers [10] do not specify an explicit IV selection algorithm, but require that the algorithm satisfy certain properties. RFC 2451 [10] states that the IV field must be chosen at random and must not use a low Hamming-distance source, such as a counter, so that ECB encryption of very similar plaintext blocks is avoided. However, the specification does not require unpredictability of an IV to an external party, and explicitly allows the common practice of using the last ciphertext block of the previous packet as the next IV.

The use of predictable IVs leads to an adaptive chosen plaintext attack, which was pointed out by Scott Fluhrer on the IPsec working group mailing list. The attack allows an attacker to break low entropy plaintext blocks using brute force, and confirm guesses of the contents of arbitrary plaintext blocks. In this paper, we analyze the preconditions and seriousness of such attacks, and provide results of practical attack experiments that confirm the vulnerability in practice.

In the rest of this paper, the term *victim host* refers to a host that performs IPsec ESP encryption and possibly ESP or AH authentication. A *victim packet* is an IPsec-protected plaintext packet whose (arbitrary) plaintext block, the *victim block*, the attacker wants to guess. The term *attack packet* refers to the IPsec-protected plaintext packet that the attacker forces the victim host to encrypt

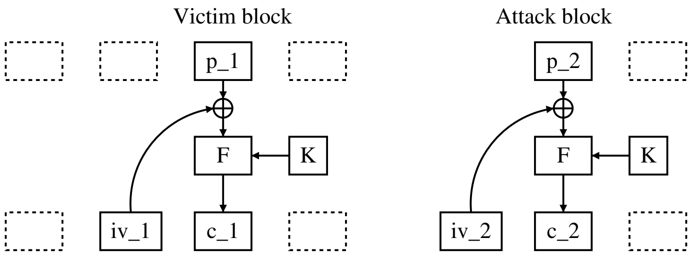


Fig. 1. The victim and attack packets

and send. The first plaintext block of the attack packet is called the *attack block*. The term *chained IV* refers to the common practice of using the last ciphertext block of the previously encrypted packet as the IV for the next encrypted packet.

2 Description of the Attack

2.1 The Attack

Figure 1 describes the ESP CBC processing of the victim and attack blocks. The attacker observes the victim packet, and then causes the victim host to encrypt and send the attack packet. The victim block may be any block of the victim packet, including the first and last plaintext blocks, while the attack block is always first in the attack packet. The victim and attack packets do not have to be adjacent in the packet stream; the victim may encrypt and send one or more packets between the victim and attack packets.

By definition of CBC, the encrypted victim block is

$$c_1 = F(K, p_1 \oplus iv_1) \ , \tag{1}$$

where $\oplus$ denotes the bitwise exclusive-or (XOR) operator, F is the block encryption function (e.g. 3DES), K is the key, p_1 is the victim block, and iv_1 is either the IV of the packet, if the victim block is the first block in the packet, or the previous ciphertext block, otherwise.

Similarly, the encrypted attack block is

$$c_2 = F(K, p_2 \oplus iv_2) \ . \tag{2}$$

The attacker chooses the attack block, p_2, as

$$p_2 = iv_1 \oplus iv_2 \oplus G \ , \tag{3}$$

where G is the attacker's guess of the victim block, p_1, and iv_2 is the attacker's prediction of the ESP initialization vector that the victim host will use to encrypt the attack packet. The encrypted attack block is then

$$c_2 = F(K, iv_1 \oplus iv_2 \oplus G \oplus iv_2) = F(K, G \oplus iv_1) \ . \tag{4}$$

If the guess G is correct, c_2 will equal c_1, confirming the attacker's guess of the contents of the victim block.

The preconditions for the attack are discussed in Sect. 3.2.

2.2 Previous Work

The security requirements for ESP IVs have been debated on the IPsec mailing list since the beginning of the IPsec working group.

A good description of this attack can be found in an e-mail sent to the IPsec mailing list by Scott Fluhrer[3]. Although the e-mail was related to AES, the attack is independent of the underlying cipher.

Phillip Rogaway pointed out that the IV generation (and security related algorithms in general) should be concretely specified to avoid cryptographically unsound implementations. He also suggested that a correlation between the IV and the first plaintext block is harmful [12]. The conclusion reached by the working group was that the IV generation has to be random to avoid correlations, but unpredictability was not required.

Later, Hugo Krawczyk pointed out a chosen plaintext attack against predictable IVs (especially chained IVs). The attack reveals the cipher key being used for encryption, but requires $O(2^n)$ memory, where n is the cipher key size in bits[4].

Philip Gladstone suggested that, although unlikely, a predictable IV might open IPsec up to a chosen plaintext attack[5]. Others commented that such an attack is not practical; the attacker cannot choose the plaintext directly because there are protocol headers before the actual plaintext, and the attacker does not have full control of the headers.

The overall consensus seems to have been that it is sufficient that the IV does not correlate with plaintext. Thus, a random IV, predictable or not, is acceptable – in particular, the common practice of IV chaining is acceptable. The attack described by Hugo Krawczyk is difficult to exploit in practice, especially against a cipher with a large keyspace, and the potential vulnerability against chosen plaintext attacks (described by Philip Gladstone and others) was also considered impractical.

3 Analysis of the Attack

3.1 Assumptions

Throughout the discussion below we assume that a cipher with a 64-bit block size is used. The attack applies to arbitrary block sizes, but the analysis details vary depending on how the block boundaries align with the protocol headers and data. Similarly, we only cover IPv4 although the attack applies to IPv6 as well.

[3] See [13], message titled "Suggested modification to AES privacy draft", January 2002.
[4] See [13], message titled "pf_key comments (predictable IVs)", January 1997.
[5] See [13], message titled "Re: I-D ACTION:draft-ietf-ipsec-skipjack-cbc-00.txt", May 1999.

3.2 Preconditions for the Attack

The attacker must be able to observe a victim packet in ciphertext form, and to extract the ciphertext block, c_1, corresponding to the victim block, p_1, and the IV used in encrypting the victim block, iv_1.

The attacker has to make a guess about the entire contents of the victim block in order to generate one attack packet. If there are n possible contents for the victim block, the attacker has to try each of the n alternatives in turn.

The attacker must be able to predict the IV, iv_2, which the victim host will use to encrypt the attack packet. The prediction can be verified from the encrypted attack packet. If the victim host uses IV chaining, this precondition means that the attacker has to capture the last ciphertext block of the packet encrypted immediately prior to the attack packet.

The attacker must be able to force the victim to encrypt and send the attack packet. Determining the desired contents of the attack block is simple, involving simply XOR. However, when the victim host processes the attack packet, the attack block corresponds to a protocol header in transport mode, and an IP header in tunnel mode. Thus, the attack block must meet any validity constraints of the header in question; otherwise the victim host will refuse to encrypt the packet.

If the attacker determines that the attack block does not meet such validity constraints, the attacker simply forces the victim to encrypt and send a dummy packet (which meets the constraints). This resets the predicted IV to a new value, and changes the corresponding attack block to a new value. The attacker then simply tries again with the new attack block. If necessary, the attacker can iterate this process indefinitely until the attack block meets the validity constraints (or the security association expires). Having obtained an attack block (dummy or not), the attacker must force the victim host to encrypt and send the attack packet, for which transport and tunnel mode require a different approach. We will analyze this precondition in more detail in Sect. 4.3.

The attacker must be able to observe the encrypted version of the attack block. If the encrypted attack block (c_2) equals the encrypted victim block (c_1), the guess in the attack packet was correct. The attacker should also be able to verify that the encrypted IPsec packet received is actually the encrypted attack packet and not some unrelated encrypted packet. If this condition is not met, the attack may yield a false negative. Verifying this condition reliably seems impossible because of encryption. However, the attacker may use e.g. timing and length of the encrypted packet as sanity checks and redo the attack if such checks fail. The attacker may also simply rely on her luck and compensate by attempting every guess several times.

The attacker must also verify that the attack packet was encrypted using the same cipher key as the victim packet. This can be done by simply verifying that the SPI fields in the two packets agree because the SPI maps statically to the cipher parameters, including the key.

3.3 Analysis of the Preconditions

Observing the victim block requires that the attacker be able to passively monitor the packets routed between the victim host and the other IPsec endpoint; the attacker may also use a routing attack to get access to the packets. If the victim uses IV chaining, predicting the IV is trivial; however, any predictable method of choosing the IV opens up the same vulnerability. Verifying the resulting encrypted attack packet is trivial. The remaining preconditions are covered in Sections 4.2 and 4.3.

The vulnerability can be confirmed by the attacker before she decides to mount an attack. The attacker can simply monitor the encrypted traffic flow and ensure her IV prediction algorithm works correctly. Also, if IKE [8] is used to set up the IPsec security associations, vendor identification payloads in the phase 1 IKE messages may provide a clue about the IV generation algorithm used by the implementation. If the attacker cannot correctly predict the IVs, she can simply give up the attack as futile without wasting resources or getting caught while attempting an active attack.

3.4 Kinds of Attacks

Brute Force Guessing. The attacker may simply iterate through a set of possible plaintexts in the victim block. Such attacks are difficult to mount if there are more than one or two octets to guess, because the amount of attack traffic easily becomes excessive.

Some potential realizations of this kind of attack:

1. The victim sends a TCP segment with a single character of e.g. password data. The attacker iterates through all the possibilities and discovers the character. (We tried this attack; see Sect. 5 and [2].)
2. The victim downloads a file from an IPsec-protected FTP site. The attacker iterates through all potential files to determine which one was downloaded. In more detail, the attacker identifies one or more plaintext blocks that are different in every potential file, and then iterates through these plaintext block possibilities.

Confirming Suspected Plaintext. The attacker may have a strong suspicion about the plaintext, and simply wants to use the vulnerability to verify her guess. In such cases, the attack is extremely efficient, and can verify a large amount of plaintext very easily.

Some potential realizations of this kind of attack:

1. The victim is sending an e-mail to a correspondent. The attacker verifies the receiver (and the sender) of the mail by predicting what the headers should (probably) look like, and by verifying these plaintext guesses. If the position of the address in the packet is uncertain, the attacker simply shifts the guess through all possible positions in the packet.

2. The victim accesses a web page. The attacker has a suspicion that the victim is accessing a certain URI, and verifies her suspicion. Note that even if the web server address is known, the attacker may be interested in knowing which URI the victim is accessing. (We verified that this attack is feasible; see Sect. 5 and [1].)
3. The attacker may routinely scan all e-mail correspondence of the victim for a few chosen words, by trying each word at every possible place in each IPsec packet that might be related to e-mail.

3.5 Effort Estimate

To uncover the contents of a single victim block requires an average of

$$N = \frac{W}{2p} \tag{5}$$

attempts, where p is the probability that the computed attack block meets the validity constraints imposed by the victim host (given a predicted IV), and W is the maximum number of possible plaintexts p_1.

Suppose that, $p = 2^{-16}$ (which is a realistic figure for IPsec tunnel mode attack; see Sect. 5) and $W = 256$, In this case, $N = 2^{23}$. At 100 packets per second, the attack requires an average of about 23.3 hours. At 1000 packets per second, the average is 2.3 hours.

When using chained IVs, the time between sending an attack packet and observing the encrypted version (containing the next IV prediction) dictates the maximum rate of attack packets. Thus, network latency plays a crucial role in the feasibility of the attack (e.g. 100 packets per second corresponds to 10 ms cycle time).

If the attacker is able to predict IVs for multiple packets in advance, latency becomes less important for feasibility. A pseudo random IV generator independent of the ciphertext (or plaintext) in previous packet(s) would allow this, but such generators do not seem to be used in practice.

4 Carrying out the Attack

4.1 Guessing the Victim Block

It is not feasible to successfully attack a completely unknown plaintext block in practice. The attacker thus needs to somehow limit the number of alternative plaintext blocks. Doing so is dependent on how the plaintext aligns with the cipher blocks. Attacking the last plaintext block of the victim packet is usually easier than attacking the other plaintext blocks, because the last plaintext is padded with a (usually) deterministic padding.

The ESP padding consists of 0...255 padding octets, followed by a “pad length” field indicating the number of such padding octets, followed by a “next header” field indicating which protocol ESP protects (the value 4, i.e. IP-IP tunnelling, is used for IPsec tunnel mode). The padding octets are used to bring the total amount of plaintext to a multiple of the cipher block size. However, the implementation is allowed to add extra padding octets in order to conceal the true length of the encrypted data. The padding octets are specified to have the form (in hex) `01 02 03`, etc, unless the cipher in question has a different requirement. [7]

In practice, all cipher algorithms for ESP use the default padding octet sequence. Since most implementations also use a minimum size padding, the entire sequence of octets following the payload data is completely deterministic.

An example: suppose that the last plaintext block contains a single data octet 0x7a, and tunnel mode is used. The last plaintext block then contains the following octets (in hex): `7a 01 02 03 04 05 05 04`. All octets except the data octet are known (assuming that the implementation uses the shortest padding sequence).

Attacking the first user data octets is, in many cases, complicated by interference from a preceding protocol header; if the first user data octets share a cipher block with the protocol header, the attacker must guess the protocol header in addition to the user data. We next cover how the user data in UDP [4] and TCP [5] may be guessed in both transport and tunnel mode.

UDP. In transport mode, the UDP payload begins at an eight octet boundary, and thus there are never any unpredictable octets (other than the unknown data) in the plaintext blocks.

In tunnel mode, the IPv4 header preceding the UDP header changes the alignment (assuming there are no IPv4 options): the first plaintext block containing UDP payload data also contains the "length" and "checksum" UDP header fields. The attacker may either try to guess both fields (which can be done with a good probability), or try to force IPv4 options that fix the alignment[6] to be used (which is difficult, because the attacker does not construct the packet). If the victim block is farther in the UDP data, such changes in alignment do not require guessing any header fields.

TCP. TCP options change the alignment of TCP user data with the cipher blocks. Similarly to UDP in tunnel mode, the attacker may compensate by guessing the TCP header fields in addition to the user data. The attacker may also try to force the victim to use suitable TCP or IPv4 (in tunnel mode) options that fix the alignment, but this is difficult because the attacker cannot directly affect the construction of the victim packet.

In transport mode without TCP options, the "checksum" and "urgent pointer" fields of the TCP header interfere with the first four octets of user data. While the urgent pointer is almost always zero, the checksum field is considerably harder to guess, because it is affected by e.g. the sequence and acknowledgement numbers, the window size, etc. A TCP header with $4+8n$ octets of options (and padding) does not interfere with the first plaintext containing data.

[6] Note, however, that some IPsec implementations do not deal correctly with IPv4 options.

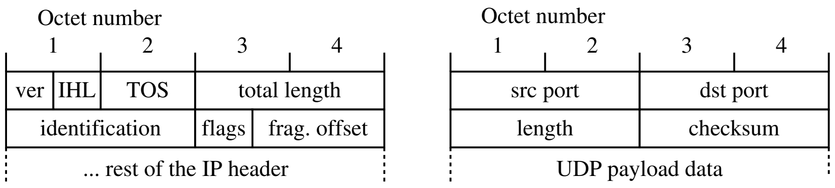


Fig. 2. Partial IPv4 header and UDP header

In tunnel mode, a TCP header without options does not interfere with the first plaintext block containing data. Thus, to attack TCP in tunnel mode, the attacker would prefer to either have no TCP options, or have an integral multiple of eight octets of TCP options.

4.2 Controlling the Attack Block

One of the preconditions of the attack described is that the attacker must be able to control the first plaintext block of the attack packet. Such control is heavily dependent on the IPsec encapsulation mode (tunnel or transport).

Transport Mode. In transport mode, the first plaintext block of the attack packet begins with the protocol header of the protocol carried inside the IP packet. Using the UDP transport protocol is the easiest method to force the victim host to encrypt and send a chosen plaintext block, because the UDP header is exactly 64 bits long (which was assumed to be the cipher block size).

The attacker has almost full control of the UDP header (Fig. 2), except for minor limitations: the length field has a minimum value (8), zero ports should not be used, and the attacker may not be able to force an arbitrary source port to be used (due to lack of privileges, for instance).

Tunnel Mode. In tunnel mode, the first plaintext block in the attack packet consists of the first eight octets in the IPv4 header [3] (Fig. 2).

The "Version" field contains four fixed bits. The "IHL" field has a value in the range 5—15; if we assume that the attacker does not use IP options, this field contains the value 5, and thus four fixed bits. The "Type of Service" field can be entirely controlled by the attacker, but may be modified by some routers; we assume that the attacker controls this field.

The "Total length" field is limited by the medium used; we assume Ethernet and thus the total length must be 20 at minimum and 1470 at maximum[7] The

[7] Because we did not want the IPsec packets to be fragmented, the maximum total length is 1500 minus IPsec overhead; the overhead consists of SPI (4 octets), sequence number (4 octets), the initialization vector (8 octets), padding, padding length and "next header" field inside ESP (2 octets, at minimum), and the ESP authenticator (we are assuming 12 octets). The resulting maximum total length is 1470.

attacker can control $\log_2 1451 \approx 10.5$ bits, while approximately 5.5 bits cannot be controlled.

The "Identification" field can be controlled fully. The "Flags" field consists of three flags: the reserved bit (set to zero), the "Don't Fragment"-bit (assumed set to zero to avoid problems), and the "More Fragments"-bit, which can be controlled. The "Fragment Offset" field can be chosen freely, as long as it is compatible with the "Total length" field[8].

In summary, with the given assumptions, there are roughly 16 bits beyond the control of the attacker.

A Note on Encrypting Fragments. Above, we assume that the IPsec implementation being attacked encrypts fragments in tunnel mode; this assumptions holds for FreeS/WAN 1.91 which was used in our tests. Some IPsec implementations first reassemble all the fragments, then encrypt, and finally fragment the resulting packet again. Attacking such implementations is more difficult because the fragment related fields cannot be freely controlled; the attack is harder (roughly) by a constant of 2^{13} (the "Fragment Offset" field).

4.3 Forcing the Victim Host to Encrypt the Attack Packet

Forcing the victim to encrypt and send the attack packet is the most difficult part of the attack. Satisfying this precondition is entirely different in tunnel and transport mode, and is also sensitive to the network topology.

Tunnel Mode. The attacker may route attack packets through the tunnel if the attacker has access to the network behind the tunnel endpoint. The common objection to this approach is that the attacker must have access to the trusted side of the tunnel, and thus there is no point in attempting this attack anyway. This argument is, however, not always valid.

The trusted network might be a large routed network. The victim and the attacker may reside in entirely different parts of the network, and the attacker might not be otherwise able to observe the victim's traffic.

Also, if the network setup allows access to an external network (through a NAT or a firewall), the attacker may be able to carry out the attack without access to the internal network. For instance, if NATted Internet access is allowed, any NAT mapping (created by the victim host by accessing the Internet) can be exploited by the attacker. The attacker can then simply forge IPv4 packets and send them using the NATted address information. If there is no NAT or a stateful firewall, the attacker can send attack packets directly without waiting for the victim to initiate an external access.

Note, however, that in such attacks, the attacker is only able to attack IPsec traffic directed towards the victim host – not traffic coming from the victim host.

[8] The combination of "Total Length" and "Fragment Offset" (converted to octets) must not exceed 65536, the size of the maximum IPv4 packet.

This is fortunate, because the traffic sent by the victim (e.g. passwords) is often more interesting than the traffic flowing in the reverse direction.

Ciphers with larger than 64-bit block size force the attacker to control more bits of the IPv4 header. For instance, a cipher with a 128-bit block size (such as AES) force the attacker to control the source IPv4 address, the TTL, and the checksum fields, among other fields.

Transport Mode. Applications running on the victim host may provide a method for sending a chosen plaintext block. Such applications could include e.g. streaming protocols, FTP, and SMTP. The victim may, for instance, have separate IPsec transport connections to both the attacker and a third host. The application on the victim host may "route" data from one connection to another at the application layer (consider, for instance, e-mail).

If the victim host is a multi-user machine, one user may be sending data using one application while the attacker is causing attack packets to be sent by using another. This approach may work for tunnel mode as well.

5 Attacks

We first verified the vulnerability manually by setting up an IPsec transport connection between Alice and Bob, and sending an UDP packet with known plaintext from Alice to Bob. We then captured the IPsec-processed ESP packet, and extracted the encrypted victim block and the IV used to encrypt it. Based on these two blocks, the known plaintext, and an IV prediction, we computed the attack block, and forced Alice to encrypt the attack packet. This was done by choosing an UDP data payload that forces the UDP header port fields, checksum field, and length field to values matching the attack block.

We then tried an attack against suspected plaintext in transport mode. Alice accesses a web page using HTTP protected by IPsec transport mode. Eve suspects that Alice is accessing a certain URI on the web server. By using the attack described in this paper, Eve verifies her guess. The attack was a success [1]. The URI of the web page being accessed was uncovered with three attack packets, confirming a guess of 24 plaintext characters[9], which is an optimal number of packets for this kind of an attack. Note that this attack is easy to carry out in tunnel mode as well.

The final attack was the most demanding one. We used a setup of three hosts, shown in Fig. 3. Alice and Bob run Linux FreeS/WAN, while Eve runs Linux with custom software able to sniff (encrypted) packets exchanged by Alice and Bob, and route forged packets through Alice in a tunnel mode attack. We used ESP with 3DES and HMAC-SHA1 for encryption and authentication, respectively.

[9] Attacking a multi-user host where several users share an IPsec security association is not new. Bellovin describes a similar attack against ESP without authentication in [14]. However, the attack described here works regardless of ESP or AH authentication.

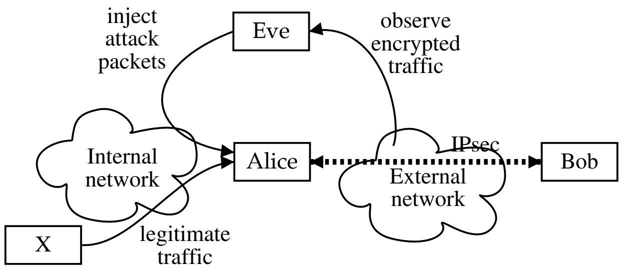


Fig. 3. Attack setup

A user of host X logs into Bob using a telnet-like protocol, protected by an IPsec tunnel. The login consists of single-character TCP segments[10]. Eve captures the encrypted packets and determines each character in turn.

We first sent a single test character through the IPsec tunnel connection, and tried the attack first using a few false guesses and then using the correct guess. This attack worked, and we were able to correctly verify desired characters of the login traffic.

Our intention was then to crack a single unknown character to obtain a practical effort estimate. This attack failed because the victim rekeyed spontaneously during the attack; we ran out of time and could not continue the attack further. Note that rekeying does not really prevent the attack – we could have continued after waiting for the victim to login again.

Even though the second part of the attack failed, the first part indicates that the vulnerability can be exploited, as long as the attacker deals with rekeying events. We were able to obtain an effort estimate in our network from the second part of the attack: the average rate of attack packets was about 1012 packets/second (0.987 milliseconds between packets). Verifying a single character guess required approximately 51821 attack packets and 51.2 seconds on average. The probability of the attack block meeting the validity constraints for IPv4 headers was approximately $2^{-15.7}$[2], which is slightly better than the pessimistic estimate of 2^{-16} (which corresponds to 65536 attack packets per guess, on average) and can be reduced further by exploiting the specific features of the victim IPsec implementation.

Given these estimates, it would take an average of 5.4 hours to uncover an eight character password consisting of ASCII characters in the range `0x20...0x7e`.

Note that our network was very simple, and thus had extremely low latency. The time required for the attack increases linearly with the latency (unless the attacker is able to predict more than one IV at a time).

6 Preventing the Attack

The change suggested by Fluhrer was to require that in addition to being random, the IV must be unpredictable to an external party. Of course, the same

[10] In real environments, the username and/or password characters might be combined into larger TCP segments.

requirement suffices to prevent the attack regardless of which cipher is used. Any algorithm for choosing IVs that can be predicted by the attacker opens up the same vulnerability; IV chaining is simply one vulnerable algorithm.

Using another ESP cipher has no effect on the attack, unless the cipher block size changes. In particular, the adaptive chosen plaintext attack described by Fluhrer is feasible even if the cipher itself resists such attacks. A larger block size makes the attack harder because there is more plaintext data to guess, and more bits to control in the attack packet.

Rekeying slows down the attacker, because an iteration of guesses against a given plaintext block cannot be continued if the key changes. The attacker can, however, wait for the victim to resend the interesting plaintext and continue the attack (of course, the attacker must somehow guess which encrypted block is a resend of the previous plaintext). Note that the attacker does not need to restart the attack from scratch; previously eliminated guesses do not have to be reconfirmed. Thus, rekeying does not protect against the attack fully.

If IV chaining is used, sending high speed data through the IPsec connection makes the attack very difficult: once the attacker has predicted the initialization vector, it may have already been used before the attacker has time to exploit the prediction. However, should the high speed traffic stop, the attacker could mount the attack immediately against any previous plaintext block, even blocks that were sent when the high speed traffic was still being sent.

ESP or AH authentication does not prevent the attack, since packets are not directly modified by the attacker but the attacker is causing the victim host to encrypt the attack blocks.

Note that the attacker does not get information that helps in breaking the encryption key, and consequently a successful attack will simply reveal the contents of a plaintext block. The attacker gains no advantage for later attacks. Knowledge of a verified plaintext-ciphertext block pair may be useful information, although such pairs are easy to guess (with a high degree of certainty) anyway.

7 Conclusions

If IVs are chosen in a predictable manner in ESP, an adaptive chosen plaintext vulnerability opens up. The preconditions of the attack are restrictive, and the vulnerability is thus difficult, but not impossible, to exploit in practice.

We demonstrated that the vulnerability can be exploited to guess single characters of TCP connections, and to verify suspected plaintext blocks, such as URIs being accessed. ESP or AH authentication does not prevent the attack.

If the victim chooses IVs using an unpredictable algorithm, the attack is effectively prevented.

References

1. Antti Nuopponen, Sami Vaarala: An Attack against IPsec Transport Mode HTTP Access, http://www.hut.fi/~svaarala/publications/espiv/webaccess.html, March 2002.

2. Antti Nuopponen, Sami Vaarala: An Attack against Single Character TCP Segments Protected by IPsec Tunnel Mode, http://www.hut.fi/∼svaarala/publications/espiv/login.html, March 2002.
3. Jon Postel: Internet Protocol. Request For Comments 760, January 1980.
4. Jon Postel: User Datagram Protocol. Request For Comments 768, August 1980.
5. Jon Postel: Transmission Control Protocol. Request For Comments 793, September 1981.
6. Randall Atkinson, Stephen Kent: Security Architecture for IP. Request For Comments 2401, November 1998.
7. Randall Atkinson, Stephen Kent: IP Encapsulating Security Payload (ESP). Request For Comments 2406, November 1998.
8. Dave Carrel, Dan Harkins: The Internet Key Exchange (IKE). Request For Comments 2409, November 1998.
9. Naganand Doraswamy, Cheryl Madson: The ESP DES-CBC Cipher Algorithm. Request For Comments 2405, November 1998.
10. Rob Adams, Roy Pereira: ESP CBC-Mode Cipher Algorithms. Request For Comments 2451, November 1998.
11. Serge Vaudenay: CBC Padding: Security Flaws in SSL, IPSEC, WTLS, ...
12. Phillip Rogaway: Problems with Proposed IP Cryptography. Internet-Draft (expired; see http://www.cs.ucdavis.edu/∼rogaway/papers/comments.html), April 1995.
13. Anon.: The IPsec mailing list (see http://www.ietf.org/ for more information; archive currently at ftp://ftp.tis.com/pub/lists/ipsec).
14. Steven M. Bellovin: Problem Areas for the IP Security Protocols. Proceedings of the Sixth Usenix UNIX Security Symposium, July 1996.

An ID Coding Scheme for Fingerprinting, Randomized *c*-Secure CRT Code

Hajime Watanabe and Takashi Kitagawa

National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2, Tsukuba, Ibaraki, 305-8568 Japan
{h-watanabe,t-kitagawa}@aist.go.jp

Abstract. In digital content distribution systems, digital watermarking(fingerprinting) technique provides a good solution to avoid illegal copying and has been studied very actively. c-secure CRT code is one of the most practical schemes for such fingerprinting since it is secure against collusion attacks even though random errors are added. But it is not sufficiently secure in the case that random errors are added. In this paper, a new ID coding scheme, *Randomized c-secure CRT code*, is presented. This new scheme improves the error tracing probabilities against collusion attacks with addition of random errors.

1 Introduction

In digital content distribution systems, it is very important to avoid illegal copying since this might causes copyright infringement. Digital watermarking techniques can be used to deter illegal copying in the following way: The recipient ID of a content is watermarked(embedded) as a fingerprint into the distributed content and if an illegal copy is found, the malicious redistributer's ID can be traced from the embedded ID.

A *collusion attack* is an attack that malicious recipients collude, compare their watermarked contents and make a content which leads to a tracing error. *c-secure code with ε-error*[1] is secure against any collusion attacks by up to c recipients with ε-error on an assumption called *Marking Assumption* but, as pointed out in [6], its code length will be long if the number of colluders c is large. Additionally, even in the case that c is small, the code length is very large. In [6], *c-secure CRT code with ε-error* was proposed and it has less code length($O(n^{1/k})$) for large n and c where k is a security parameter. This code can be transformed into a random error resilient code by redefining c(for detail, see [6]) even though, by colluders(malicious recipients), random errors are added to the content.

In content distribution system, if a tracing error(an innocent recipient is detected as a malicious recipient) probability is not sufficiently small, the system might not be used widely. Unfortunately, c-secure CRT Code cannot achieve the small probability without increase of codeword's length if random errors are added. To make the tracing error probabilities smaller, an improved algorithm called A_3 for tracing malicious recipients of c-secure CRT code was proposed[9]. In this tracing algorithm, a new threshold, w_{th}, is used and a new procedure which checks neighbor blocks of a subcodeword detected from illegal copy is added to distinguish random errors and modifications by collusion attacks with addition of random errors. Differently from coding theory, the environment in which the coding scheme is used is not static in fingerprinting coding scheme. In other words, colluders can design attacks by using the knowledge of coding scheme, encoding and tracing (decoding) algorithms. In this paper, a new such collusion attack with addition of random errors is introduced. We show that even the encoding algorithm of c-secure CRT Code and the tracing algorithm A_3 does not achieve sufficiently small tracing error probability by computer simulations.

Next, we analyze why the tracing errors occurred and we propose a new coding scheme, *Randomized c-secure CRT Code* by using the result of the analysis. We evaluate the tracing error probability of our new scheme by computer simulation against the new attack and show the effectiveness of our new scheme. Of course, Randomized c-secure CRT Code is also secure against the attack considered in [6],[9]. The length of codewords is the same of c-secure CRT Code. These results indicate that Randomized c-secure CRT Code is very responsible for secure content distribution systems.

In Section 2, we explain the construction of c-secure CRT Code, its tracing algorithm and improved tracing algorithm A_3 briefly. In Section 3, we introduce an effective attack for c-secure CRT Code and discuss why the attack is effective. We propose an improved coding scheme, Randomized c-secure CRT Code, which is secure against the new attack in Section 4 and show the security in Section 5. Section 6 concludes this study.

2 Preliminaries

2.1 c-Secure CRT Code and Its Tracing Algorithm

In this section, we briefly explain the construction and the tracing algorithm of c-secure CRT code with ε-error [6]. Let n be the number of users in the distribution system. A user ID u is chosen from $\mathbb{Z}_n$ and the user ID is expressed by a set of residues by the Chinese Remainder Theorem(CRT). Each residue is encoded by $O(n)$ n-secure code [1]. The code is a shortest code for small n. c-secure CRT code is a concatenated code of these inner codes.

Modulus. Let ε_1 and ε_2 be non-negative numbers satisfying $0 < \varepsilon_1 < 1, 0 < \varepsilon_2 < 1$ and $(1-\varepsilon_1)(1-\varepsilon_2) > 1-\varepsilon$ where ε is the upper bound of the tracing error probability. Let l, k' and k be positive integers such that $\lfloor 2k'/c \rfloor = k + l$. Let $p_0, p_1, \cdots, p_{k'-1}$(without loss of generality, we assume that $p_0 < p_1 < \ldots < p_{k'-1}$) be a pairwise relative prime integers and satisfy $\prod_{i=0}^{k-1} p_i \geq n$(note that $k > k'$, and small k primes, $p_0, p_1, \cdots, p_{k-1}$, are sufficient). l is chosen such that satisfying the following condition:

$$[1 - \prod_{i=0}^{l-1}\{1 - (1 - \frac{1}{p_{k+i}})^c\}]^{{}_{k'}C_{k+l} \times 2^{kl}} \geq 1 - \varepsilon_2.$$

We call them(p_is) moduli and define an average of the moduli $\bar{p} = \sum_{i=0}^{k'-1} p_i/k'$.

Residues. For a user ID u, an integer $r_i \in \mathbb{Z}_{p_i}$ such that $r_i \equiv u \pmod{p_i}$ is a residue of u modulo p_i. if at least k distinct residues are given, the user ID u can be computed from the residues by the Chinese remainder theorem.

Inner Code. For each moduli p_i, the inner code $\Gamma_0(p_i, t)$ is defined in [1]. The codeword $w_i^{(j)}$ of $\Gamma_0(p_i, t)$ is defined as follows:

$$w_i^{(j)} = \underbrace{00\cdots0}_{t \times j}\ \underbrace{11\cdots1}_{t\times(p_i-j-1)} \text{ for } j \in \mathbb{Z}_{p_i}.$$

Here, t is the smallest integer satisfying $t \geq -\log_2[1 - (1 - \varepsilon_1)^{\frac{1}{2k'}}]$. We call the t bit strings in the inner code a block. A codeword $w_i^{(j)}$ consists of j "0" blocks and $p_i - j - 1$ "1" blocks.

***c*-Secure CRT Code.** c-secure CRT Code is a concatenated code of the residues which are encoded by the inner code. This code is denoted by $\Gamma(p_0, p_1, \cdots, p_{k'-1}; n, t)$. A codeword of c-secure CRT Code $W^{(u)}$ is defined as follows:

$$W^{(u)} = w_0^{(r_0)} \| w_1^{(r_1)} \| \cdots \| w_{k'-1}^{(r_{k'-1})} \text{ for } u \in \mathbb{Z}_n$$

where $r_i \equiv u \bmod p_i, 0 \leq i < k'$. The code length of c-secure CRT Code is

$$\sum_{i=0}^{k'-1} p_i t = \bar{p}k't.$$

Every $k(< k')$ inner codes are sufficient to detect the ID u if the codeword is not attacked since $\prod_{i=0}^{k-1} p_i \geq n$(see Modulus and Residues).

Tracing Algorithm. The tracing algorithm of c-secure CRT Code[6] consists of the following five steps.

STEP 1: Check the content and detect the embedded information x. the length of x is $\bar{p}k't$.

STEP 2: Divide x into k' words as follows:

$$x = x_0 \| x_2 \| \cdots \| x_{k'-1}$$

where length of x_i is $t(p_i - 1)$.

STEP 3: For each x_i, apply the following algorithm:

```
input $x$;
for ($min = 0$; $min < p_i - 1$; $min$ ++)
    if ($H_{min}(x) > 0$) break;
for ($max = p_i - 1$; $max > min$; $max$ −−)
    if ($H_{max-1}(x) < t$) break;
output $min$ and $max$;
```

Here, $H_{min}(x)$ is the Hamming weight of the min-th block of x. The outputs of the algorithm min and max are denoted by $r_i^{(-)}$ and $r_i^{(+)}$, respectively, where $r_i^{(-)} < r_i^{(+)}$.

STEP 4: Count numbers $\mathcal{D}(u)(0 \leq \mathcal{D}(u) \leq k')$ exhaustively for each user u. $\mathcal{D}(u)$ is the number of congruent equations which u satisfies with the residue pairs, defined as follows:

$$\mathcal{D}(u) = |\{i \in \mathbb{Z}_{k'} | (u \equiv r_i^{(-)} \pmod{p_i}) \vee (u \equiv r_i^{(+)} \pmod{p_i})\}|.$$

$\mathcal{D}(u)$ is called a *degree* of the residue pairs at u.

STEP 5: Define $D_{th} = k + l$. If the condition $\mathcal{D}(u) > D_{th}$ holds, output the user u as a member of the coalition.

2.2 Improved Tracing Algorithm A_3

The above tracing algorithm is sufficiently secure against collusion attacks by up to c malicious recipients. However, if random errors are also added to the content, tracing errors occur in not small probability. By redefining c(consequently increasing the codeword's length), the code can be transformed to a random error resilient one but it is not so effective. To overcome this problem, three improved tracing algorithms were proposed [9]. The best algorithm of them is called A_3. This algorithm is almost the same algorithm and the only difference is STEP 3.

In this algorithm, it is assumed that the random errors are added to the content with the probability e(note that e means the rate that bits of the codeword are changed), and the server(who checks the embedded ID) can compute e in some way. From the probability e, the server determine a new threshold w_{th} as follows: The blocks of the detected data can be classified into tree types, a collusion attack is executed to blocks(Type 1), only random errors are added to all "0" blocks(Type 2), and only random errors are added to the all "1" blocks(Type 3). The probability distribution of the Hamming weights on type 1, type 2 and type 3 are defined in the following way:

$$p_{\mathit{Type1}}(w) = \binom{t}{n}\left(\frac{1}{2}\right)^t,$$
$$p_{\mathit{Type2}}(w) = \binom{t}{w}e^w(1-e)^{t-w},$$
$$p_{\mathit{Type3}}(w) = \binom{t}{w}e^{t-w}(1-e)^t.$$

Let w_{th} be a maximum integer which satisfy $p_{\mathit{Type2}}(w_{th}) > p_{\mathit{Type1}}(w_{th})$. In other words, w_{th} is the maximum Hamming weight which the probability that the block was created by adding random errors to all "0" block is greater than the probability that the block was created by the collusion attack. Finally, the server chooses a small positive integer ad_{th}. The choice of ad_{th} is ad hoc. The STEP 3 of A_3 is as follows:

STEP 3 of Algorithm A_3:

```
input $x$;
for ($min = 0$; $min < p_i - 1$; $min++$)
    if ($(H_{min}(x) > w_{th}) \wedge \cdots \wedge (H_{min+ad_{th}-1}(x) > w_{th})$)
        break;
for ($max = p_i - 1$; $max > min$; $max--$)
    if ($(H_{max-1}(x) < t - w_{th}) \wedge \cdots \wedge (H_{max-ad_{th}+2}(x) < t - w_{th})$)
        break;
output $min$ and $max$;
```

3 Attacks for *c*-Secure CRT Code

3.1 Attacks

Since *Marking Assumption*[1] is supposed, for each detected places, there exist two values corresponding to "0" and "1". For each modification of detected places, the only possible thing to do for colluders is to select one value from these two values.

On the evaluation of tracing algorithms in [6],[9], only the following collusion attack with addition of random errors was considered since *Additional Marking Assumption*[6] is supposed:

Uniform Selection Attack

1. Compare and detect the places in which fingerprint embedded.
2. Make a possible modified codeword using the knowledge of detected places. For each modification of detected places, the selection of each value is done randomly and independently.
3. Add random errors to the result content.

But there exists a collusion attack with addition of random errors whose success probability is higher than the above attack. This attack turns the new threshold w_{th} to evil ends by using the knowledge of the encoding and the tracing algorithms. The attack is executed in the following way:

Threshold Based Attack

1. Compare their contents and detect the places fingerprint embedded.
2. Select two colluders who have most different contents.
3. Decide a error probability calculated in tracing and calculate the corresponding threshold w_{th}.
4. Make a possible modified codeword from two contents of the selected colluders. For each modification of detected places, the selection of each value is based on the threshold. The ratio of selecting one's value or the other's value is
$$w_{th} : t - w_{th}$$
where t is a parameter and w_{th} is the introduced new threshold in [9].
5. Add random errors in the probability decided in STEP 3 to the result content.

In this tracing algorithm, w_{th} is calculated from the random error probability. The probability is calculated in tracing. To succeed in the attack, colluders must know the probability in advance. Colluders do not have to keep the probability secret in this case. By using the knowledge of the estimation way of the probability in tracing or simply adding random errors, colluders can easily manipulate the threshold calculated in advance. That is why colluders can do this attack.

We show the probabilities of error tracing against Threshold Based Attack plotted by $\diamond$ and those in which no one can be detected plotted by $+$ in Figure 1 by simulation(the value of each point is the average of 1000 trials). The parameters used are the same of [9]: $c = 15, m = 52, k = 2, t = 25, p_0 = 100, n = 1.0 \times 10^4, l = 5, L = 2.77 \times 10^6, ad_{th} = 3$ where

c	max number of colluders supposed here
k, m, l	parameters (see Modulus)
t	block length(see Inner Code)
p_0	smallest moduli($p_1 = 101, \ldots, p_{m-1} = 359$)
n	number of users in the system
L	code length
ad_{th}	window size in A_3

and $ID = 0, 1, \ldots, 49$ are not used as weak IDs[6]. The discontinuities in Figure 1 mean the changes of the threshold value w_{th}. Note that, against Uniform Selection Attack, the average error probabilities of tracing residues in the inner codes are not so high and the average probabilities of error tracing are 0[9].

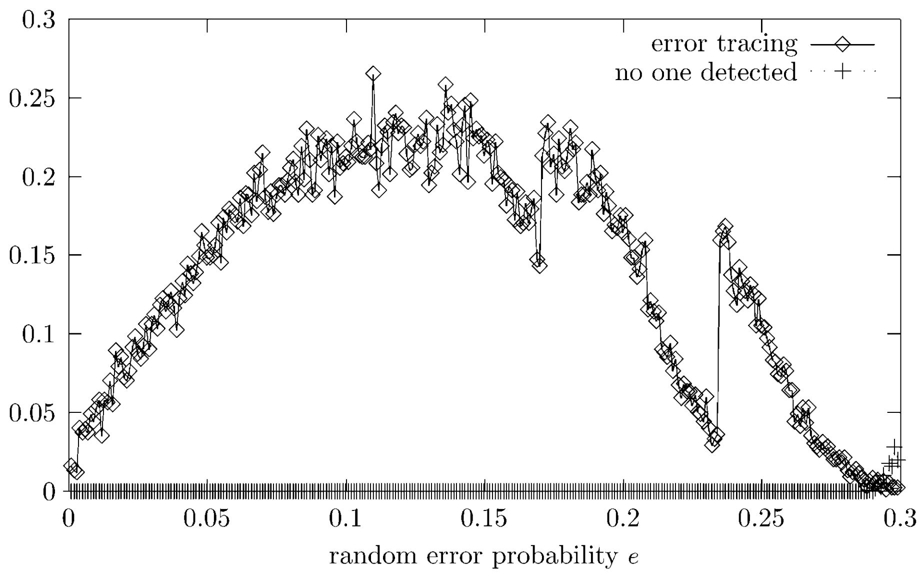


Fig. 1. Tracing Error Probabilities of c-secure CRT Code

3.2 Discussion

In this section, we discuss why Threshold Based Attack is effective for A_3 tracing algorithm.

Threshold Based Attack makes the random errors and the modifications indistinguishable since the ratios of "0" and "1" in the inner codes are similar with not small probability. To succeed in the attack, colluders must furthermore make a modified codeword of which the same innocent user can be detected from sufficiently many inner codes. This can occur with not small probability.

We will show the fact by using examples. Let two adjacent IDs be 123 and 124, and let $\{p_i\} = \{\ldots, 57, \ldots, 101, \ldots\}$. In this case, the inner codes of 123 and 124 for modulus 57 and 101 are in Table 1. Residues of adjacent IDs are also adjacent with high probability. Therefore corresponding inner codes of moduli are adjacent with high probability, too.

Table 1. Relations between IDs and Residues

ID	residue of modulus 57	residue of modulus 101
123	9	22
124	10	23

Consider the case that two users collude and their IDs are 123 and 150. Look at the Table 2. To succeed in the attack for the inner code of modulus 57, the modification of leftmost(or rightmost) X block must be considered as the result of addition of random errors in this inner code. The probability that they make the modified inner code such that an innocent user ID $u(124 \le u \le 149)$ is detected is not small. The reason is that, in 4 of Threshold Based Attack, the ratio of selecting one's value or the other's value is $w_{th} : t - w_{th}$ and the ratio of "0" and "1" in the block is also similar with high probability. This means that the number of "1"("0") in leftmost(rightmost) X does not exceed w_{th} and the modification of leftmost(or rightmost) X is considered as that with not small probability.

Table 2. A Modification of an Inner Code

ID	inner code of modulus 57
123	$\underbrace{00\cdots0}_{t\times9}\underbrace{11\cdots1}_{t\times47}$
150	$\underbrace{00\cdots0}_{t\times36}\underbrace{11\cdots1}_{t\times19}$
collusion attacked data	$\underbrace{00\cdots0}_{t\times9}\underbrace{XX\cdots X}_{t\times29}\underbrace{11\cdots1}_{t\times19}$

For each inner code, the same discussion can be applied. And since adjacent IDs relate their residues and their inner codes, it is not very difficult for colluders to create sufficiently many such inner codes that the same ID(for example, ID 124) can be detected.

4 Randomized c-Secure CRT Code

As the result of analysis in the above discussion, we proposed a new ID coding scheme, *Randomized c-secure CRT Code.* Our encoding algorithm is based on c-secure CRT Code. The only difference is to add random permutation processes $P_i(0 \le i \le k'-1)$ for breaking the correlation between adjacent IDs and their inner codes. Our tracing algorithm is based on A_3. The only differences are the processes of treating the additional random permutations.

Random Permutations. For each modulus $p_i(0 \le i \le k'-1)$, the random permutation P_i of numbers from 0 to $p_i - 1$ is defined. The k' tables of these permutations are created and stored by the server who embeds and detects fingerprints.

Randomized c-Secure CRT Code. Randomized c-secure CRT Code is a concatenated code of the residues which are encoded by the inner code. This code is denoted by $\Gamma_R(p_0, p_1, \cdots, p_{k'-1}; n, t)$. A codeword of Randomized c-secure CRT Code $W_R^{(u)}$ is defined as follows:

$$W_R^{(u)} = w_0^{(P_0(r_0))} \| w_1^{(P_1(r_1))} \| \cdots \| w_{k'-1}^{(P_{k'-1}(r_{k'-1}))} \text{ for } u \in \mathbb{Z}_n$$

where $r_i \equiv u \bmod p_i, 0 \leq i < k'$. The code length of Randomized c-secure CRT Code is also $\bar{p}k't$.

Tracing Algorithm. The tracing algorithm of Randomized c-secure CRT Code consists of the following five steps. The differences between this and A_3 are only STEP 4 and 5.

STEP 1: Check the content and detect the embedded information x. the length of x is $\bar{p}k't$.

STEP 2: Divide x into k' words as follows:

$$x = x_0 \| x_2 \| \cdots \| x_{k'-1}$$

where length of x_i is $t(p_i - 1)$.

STEP 3: For each x_i, apply the following algorithm:

```
input $x$;
for ($min = 0$; $min < p_i - 1$; $min++$)
    if (($H_{min}(x) > w_{th}$) $\wedge \cdots \wedge$ ($H_{min+ad_{th}-1}(x) > w_{th}$))
        break;
for ($max = p_i - 1$; $max > min$; $max--$)
    if (($H_{max-1}(x) < t - w_{th}$)$\wedge \cdots \wedge$ ($H_{max-ad_{th}+2}(x) < t - w_{th}$))
        break;
output $min$ and $max$;
```

STEP 4: Count numbers $\mathcal{D}_R(u)(0 \leq \mathcal{D}(u) \leq k')$ exhaustively for each user u. $\mathcal{D}_R(u)$ is the number of congruent equations which u satisfies with the residue pairs, defined as follows:

$$\mathcal{D}_R(u) = |\{i \in \mathcal{Z}_{k'} | (P_i(u) \equiv r_i^{(-)} \pmod{p_i}) \vee (P_i(u) \equiv r_i^{(+)} \pmod{p_i})\}|.$$

STEP 5: Define $D_{th} = k + l$. If the condition $\mathcal{D}_R(u) > D_{th}$ holds, output the user u as a member of the coalition.

5 Security

We show the security of Randomized c-secure CRT Code by showing the probabilities of error tracing against Threshold Based Attack. The probabilities are plotted by $\diamond$ in Figure 2 by computer simulation(the value of each point is the average of 10000 trials). The parameters used are also the same of [9]. Note that, since a random permutation is used in our proposed code, we do not have to consider weak IDs any more. Since Randomized c-secure CRT Code uses random

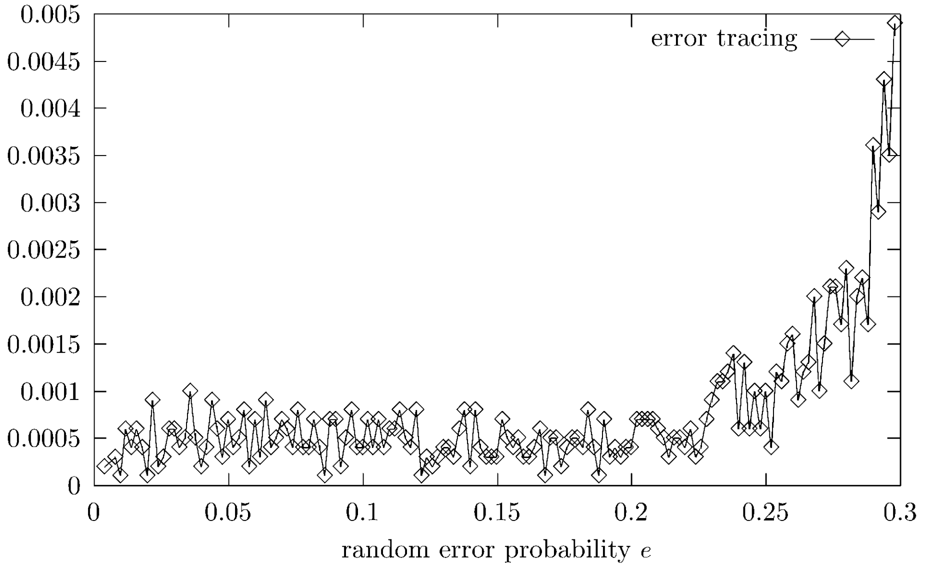


Fig. 2. Tracing Error Probabilities of Randomized c-secure CRT Code

permutations described in previous section, the residues of adjacent IDs are not adjacent in their inner codes. For this reason, it is more difficult to make an attacked codeword that the same innocent user is detected from its inner codes. Consequently, the tracing error probabilities against Threshold Based Attack are drastically improved(less than 1/200 at many points).

Of course, our new code is also secure against Uniform Selection Attack since our modification does not affect tracing errors. It might be more secure than c-secure CRT Code but, because the previous work has achieved sufficient security against the attack, we omit the results.

6 Conclusion

In this paper, we have proposed a collusion attack with addition of random errors called Threshold Based Attack. This attack is effective in breaking the security of c-secure CRT Code since it uses the knowledge and the characteristic of the coding scheme.

To solve the problem, we have proposed a new ID coding scheme called Randomized c-secure CRT Code. Since this code does not have such characteristic, it is also secure against the new attack. Randomized c-secure CRT Code is one of the most suitable ID coding schemes for secure content distribution systems. On designing an ID coding scheme, it is very important to consider not only the simple attack but also attacks which uses the knowledge of the coding scheme like Threshold Based Attack.

References

1. D. Boneh and J. Shaw: "Collusion-Secure Fingerprinting for Digital Data," *Proc. of CRYPTO'95*, LNCS963, pp.452–465, Springer-Verlag, 1995.
2. J. Yoshida, K. Iwamura and H. Imai: "A Coding Method for Collusion-Secure Watermark and Less Decline," *Proc. of SCIS'98*, 10.2.A, 1998(In Japanese).
3. D. Boneh and J. Shaw: "Collusion-Secure Fingerprinting for Digital Data," *IEEE Transactions on Information Theory*, vol.44, no.5, pp.1897–1905, 1998.
4. H-J. Guth and B. Pfitzmann: "Error- and Collusion-Secure Fingerprinting for Digital Data," *Proc. of IH'99*, LNCS1768, pp.134–145, Springer-Verlag, 2000.
5. H. Muratani: "Collusion Resilience of Digital Watermarking," *Proc. of SCIS2000*, C-06, 2000(In Japanese).
6. H. Muratani: "A Collusion-Secure Fingerprinting Code Reduced by Chinese Remaindering and Its Random-Error Resilience," *Proc. of IH 2001*, LNCS2137, pp.303–315, Springer-Verlag, 2001.
7. K. Yoshioka and T. Matsumoto: "Random-Error-Resilient Tracing Algorithm for Collusion-Secure Fingerprinting Code," Technical Report of IEICE, ISEC2001-52, pp.247–254, 2001(In Japanese).
8. H. Muratani: "Combinatorial Outer Codes for c-Secure CRT Code," *Proc. of SCIS2002*, Vol.2, pp.1009–1014, 2002(In Japanese).
9. K. Yoshioka and T. Matsumoto: "Random-Error-Resilient Tracing Algorithm for Collusion-Secure Fingerprinting Code (Part2)," *Proc. of SCIS2002*, Vol.2, pp.1021–1026, 2002(In Japanese).

A Robust Block Oriented Watermarking Scheme in Spatial Domain

Tanmoy Kanti Das[1] and Subhamoy Maitra[2]

[1] Computer Vision and Pattern Recognition Unit, Indian Statistical Institute
203, B T Road, Calcutta 700 108, India
das_t@isical.ac.in

[2] Applied Statistics Unit, Indian Statistical Institute
203, B T Road, Calcutta 700 108, India
subho@isical.ac.in

Abstract. In this paper we present an invisible spatial domain watermarking technique. The technique divides the image into n small blocks and the intensity of some of these blocks are modified depending on the key, which is a secret random binary string of length $m < n$. Given an attacked image, the recovery process generates a bit pattern of length m and we show that from this pattern it is possible to get back the exact key using either standard correlation measure or error correcting codes. The recovery process employs window matching technique on the attacked image, which has been found to work successfully. Our method can survive common image transformations and intentional attacks both in spatial and frequency domain. Most importantly, it also survives the nonlinear geometric attacks, e.g., Stirmark. To the best of our knowledge, here we experiment with an exhaustive set of attacks which has not been provided elsewhere.

Keywords: Information Hiding, Digital Watermarking, Gray Scale Image, Stirmark, Cryptanalytic Attacks.

1 Introduction

Now a days most of the multimedia contents are in digital form. With the increased use of Internet, the digital objects can be distributed very efficiently and rapidly over the net. However, the question of copyright protection becomes an issue to be taken care of. Watermarking technologies are thus becoming natural choice of the content creators in this direction. This area of research is not only used in establishing the ownership but also being exploited to identify the copyright infringer.

Most of the existing invisible watermarking techniques exploit the characteristic of human visual system and except a few, most of them are correlation based. The embedding process works in a manner so that very little visible modification occurs to image. These modifications correspond to the watermarking information and should be robust enough to withstand any kind of attack. Correlation between the recovered information and embedded information are used as the measure of confidence in the detection process. The watermarking strategies are employed in the spatial domain as well as different transform domains like Discrete Cosine Transform, Fast Fourier Transform, Wavelet Transform etc. See [2,3,6,7,8,9,10,11,15,17] and references in these papers for more details.

In this paper we present a block based spatial domain watermarking scheme. Many spatial domain watermarking techniques find the perceptually significant regions of the image and embeds the watermark in those places [8,9]. Our algorithm is independent of such assumptions. We use slight modification of pixel values to insert the watermark. One early spatial domain watermarking strategy, Patchwork [1], also uses modification of pixel values to embed the watermark. However, our method is much more involved and robust than Patchwork.

Let us now briefly outline the requirements for a robust watermarking scheme.

1. The watermark should remain secret and should only be accessible to owner, even if the scheme is available in the public domain. Moreover, it should preserve the perceptual quality of the image.
2. It must withstand common signal processing operations both in frequency and spatial domain such as lossy compression, scaling, cropping, rotation, low pass filtering etc. and their combinations. More importantly, the method should provide robustness against nonlinear geometric attacks (distortion attacks) such as Stirmark [13,14,18].
3. It should prevent any intentional cryptanalytic attacks. Moreover, more than one buyer should not be able to collude and remove the watermark.

Methodologies to test the robustness of watermarking schemes are now well known [15,13,9] and our technique survives these tests. To the best of our knowledge, here we provide simulation of an exhaustive set of attacks which has not been experimented elsewhere. In particular the Stirmark [13,14,18] program provides remarkable success in disabling watermark recovery process. We show that our method can survive this attack too. Our method divides the image into several blocks of size $\beta \times \beta$. To embed a bit value 0 (or 1) into a block we decrease (or increase) the pixel values of that block by certain amount δ. The value of δ is calculated dynamically depending on the block at hand. Extraction procedure employs several techniques to resist sophisticated watermark removal and disabling attack. To extract a bit from a block we compare the corresponding block of watermark and original image. Prior to the comparison window matching algorithm is employed for each block to ascertain (and rectify) whether the watermarked image is subjected to any nonlinear geometric attack.

In the next section (Section 2) we explain the algorithms in detail for both the watermark insertion and watermark recovery strategies. In Section 3 we present experimental results to show that our scheme provides necessary robustness.

2 Watermarking Scheme

The embedding process is executed for each buyer. The method is as follows.

1. Read the host image into I.
2. Divide I into number of blocks of size $\beta \times \beta$. Let $n = \frac{ht(I) \times wd(I)}{\beta \times \beta}$. By $ht(I), wd(I)$, we mean the height and width of the image in pixels.

3. Take a permutation π of n integers $0, \ldots, n-1$ (which is same for all the buyer) and a binary pattern k of length $m < n$. Insert k in the database along with the buyer information. This database is kept secret with the owner.
4. Watermark insertion in I is done as follows. For each of the selected blocks b_{π_j}, $0 \leq j \leq m-1$, we do the following operations.
   (a) Calculate minimum ψ_l and maximum ψ_h of the intensities in the block.
   (b) Calculate a parameter δ for the block as $\delta = \max(\mu, \lceil \alpha * (\psi_h - \psi_l) \rceil)$, where μ is an integer constant with very low value (say $1 \leq \mu \leq 3$) and α is a small real value (say $0.05 \leq \alpha \leq 0.10$).
   (c) If $k_j = 1$, we increase every pixel of the chosen block by δ, otherwise we decrease it by δ.

From the nature of algorithm it is clear that in the flat region of the image, where ψ_h, ψ_l differ very little, value of $\delta = \mu$ but in case an edge is present ψ_h, ψ_l will differ by large amount thus value of δ will be much higher than μ. Thus for an attacker it becomes difficult to change the value of each block by random amount to defeat the watermarking process, without damaging the edges. Because any small change in wrong direction will destroy the edges. At this point let us discuss what is the optimal value of m. It is well known that less the watermarking payload (number bits embedded, i.e., m) more robust it is. It is experimentally found that value of m should lie between $\frac{n}{4}$ and $\frac{3n}{4}$ to resist rewatermarking as well as block based attack successfully.

Let us now calculate the key space. As already mentioned, the number of available block is n and length of the binary pattern to embed is m where $m < n$. So we have m different blocks chosen out of n blocks keeping the order of permutation. The permutation of n blocks taken m at a time is ${}^{n}\mathcal{P}_{m}$. Now the number of binary patterns considering m places is 2^m. Hence the total number of possible keys is ${}^{n}\mathcal{P}_{m} \times 2^m$. Note that we here use $n = 4096$ and $m = 2660$ for experiment. This gives a key space of size ${}^{4096}\mathcal{P}_{2660} \times 2^{2660}$. It is very clear that the key space is of huge size. Note that we do not need these many keys in any practical system. Thus, we can choose the bit pattern of length m with proper care. In fact it is better to select these m bit patterns in such a manner so that the Hamming distance (the number of places in which two same length binary strings differ) between any two keys is large. This can be done by using error correction codes [12].

The recovery process requires both the watermarked (may be attacked) image and the original host image with a complete read access to the buyer key database. Watermark is recovered by comparing accumulated intensity of all pixels in each block of host image with corresponding block of watermarked image. The comparison process is very much sensitive to small affine changes which cannot be reliably corrected. To overcome this problem we employ window matching algorithm. Let us now describe the window matching algorithm in detail.

Input : Coordinate (x, y) and window size parameters β, σ. **Output** : $\Gamma_{x,y}$.

1. Consider the original image I and the attacked watermarked image $I^{\#}$.
2. Consider a window of size $\beta \times \beta$ whose top left pixel is at location (x, y). This we consider as location of the window also. Let us denote such windows as $w(x, y)$ and $w^{\#}(x, y)$ respectively for the images I and $I^{\#}$.

3. Consider a larger window size of $(\beta + c) \times (\beta + c)$ pixels, where $\sigma = \beta + c$ at location $(x - \frac{c}{2}, y - \frac{c}{2})$ from I which we denote as $W(x - \frac{c}{2}, y - \frac{c}{2})$.
4. There are $(c + 1)^2$ number of different smaller windows $w(x + i, y + j)$ for $i = -\frac{c}{2}$ to $\frac{c}{2}$ and $j = -\frac{c}{2}$ to $\frac{c}{2}$ inside $W(x - \frac{c}{2}, y - \frac{c}{2})$.
5. Calculate sum of absolute differences of the $\beta \times \beta$ pixel values between each pair $w^{\#}(x, y)$ and $w(x+i, y+j)$ for all $i = -\frac{c}{2}$ to $\frac{c}{2}$ and $j = -\frac{c}{2}$ to $\frac{c}{2}$. Denote this by $\Delta_{i,j}$.
6. Consider the coordinate $(x + i', y + j')$ for which $\Delta_{i',j'} = \min_{i,j} \Delta_{i,j}$.
7. Report the sum of differences of the $\beta \times \beta$ pixel values between the pair $w^{\#}(x, y)$ and $w(x + i', y + j')$. This we denote as $\Gamma_{x,y}$.

The importance of the window matching algorithm becomes evident in the case of nonlinear geometric attacks such as Stirmark [18]. See the discussion in Subsection 3.3 for experimental results which shows how the robustness in the watermark recovery scheme is attained using the window matching technique. Next we present the complete watermark recovery algorithm.

1. Read the host image into I and the watermarked (possibly attacked) image into I'.
2. A preprocessing step on I' is executed.
   (a) In case of cropping we will compensate the cropped part by glueing the cropped parts from existing I to I'.
   (b) In case of resizing, rotation and translation, we bring back the image to the original size and form using standard affine transformations [5].
3. Construction of k''.
   (a) Take the n length permutation π.
   (b) We normalize the intensity of I' in following manner.
      i. From π, it is noted which of the blocks are selected for modification during watermark embedding. We first concentrate on the rest of the $n - m$ blocks.
      ii. Now we run the window matching algorithm corresponding to each block (each block is identified by the coordinate of its top left corner) of size $\beta \times \beta$. We also consider the larger window of size $\sigma \times \sigma$. For our experiment, we take $\beta = 4, \sigma = 8$.
      iii. We calculate $\nu = \lfloor \frac{\sum \Gamma_{x,y}}{(n-m) \times \beta \times \beta} \rfloor$, where the summation is over all the $n - m$ blocks not included in permutation π.
      iv. For all the pixels of I' (irrespective of whether they are in the watermarked blocks or not) $p' = p' - \nu$.
   (c) Now we construct a binary string k'' of length m as follows.
      i. Given π, we know which of the blocks are modified during watermarking.
      ii. Let us consider the block corresponding to π_j at location (x, y). For this block we run the window matching algorithm to get $\Gamma_{x,y}$.
      iii. If $\Gamma_{x,y} > 0$, we take $k''_j = 1$, if $\Gamma_{x,y} < 0$, we take $k''_j = 0$, and if $\Gamma_{x,y} = 0$, we take $k''_j = 0$ or 1 depending on coin tossing.

4. This step is executed for each of the records present in the database. Read a binary pattern k'. Now we fix a threshold γ (it should be greater than 0.5). If $\frac{M(k',k'')}{m}$ is less than γ, we reject k'. If $\frac{M(k',k'')}{m}$ is greater than γ, we accept k' as the correct key k. By $M(s_1, s_2)$, we mean number of places two same length binary strings s_1, s_2 match. (Note that the value of $\frac{M(k',k'')}{m} \times 100$, i.e., the percentage, is called the similarity factor.)

Note that in the step 4 of the recovery algorithm, we traverse through the complete database. In fact, this is not required if we use error correcting codes [12]. In that case from k'' we can directly get back k using the decoding algorithm of the respective error correcting code and the search over the complete database is not required. Let us now relate the the value γ with the error correcting codes. Consider the error correcting codes of length m, such that minimum distance between any two codes is at least d. We know that even if there are at most $\frac{d}{2}$ errors in the bit pattern of length m, then also it is possible to get back to the original code word. Given γ, we calculate d in terms of m as follows. We need, $m - \frac{d}{2} = \gamma m$, i.e., $d = 2(1 - \gamma)m$. Note that if we take $\gamma = 0.6$ and $m = 2660$, then d becomes 2128. This means for the buyer key, we need binary strings of length 2660 such that the Hamming distance between any two strings is at least 2128. By Hamming distance between two same length strings s_1, s_2, we mean number of places s_1, s_2 do not match. In the experiments, we present our results in terms of similarity factor.

Given the complex nature of the scheme it seems hard to analyse it theoretically and hence the robustness is demonstrated experimentally in the next section.

3 Experimental Results

In this section we present the experimental results to substantiate our claims. We start the processing on a watermarked image I' generated from the original image I, according to the embedding algorithm described in Section 2. In our experiment we process the images in TIF format. First we refer to the original image and its watermarked version. It is clear that there is no perceptual distinction between the original and watermarked images (see Figure 1). In the watermark embedding algorithm (step 4b) we use the values $\alpha = 0.05$ and $\mu = 1$. Note that these are the lowest possible values of α, μ for the range we propose in the watermark embedding algorithm. This provides the best possible scenario for the attacker. Even then the experiments reveal the robustness of the scheme. Note that the PSNR value [9, Page 112] of the watermarked image with respect to the original image is as high as +42.2 db for $\alpha = 0.05, \mu = 1$ (see Figure 1). For the maximum possible values in the range, i.e., $\alpha = 0.10, \mu = 3$, we find the PSNR value of +35.8 db with respect to the original image. This also keeps the perceptual quality of the watermarked image indistinguishable from the original image. In watermark recovery process value of $\gamma = 0.6$ (in terms of similarity factor it is 60%) is used.

Fig. 1. The original and the watermarked image.

3.1 Transformations and Attacks in Spatial Domain

In Table 1 we present the result related to rotation. The image I' is rotated at different angles from 10^o to 45^o to get I''. Then the image is rotated back as I'''. It is clear that in this processing some portion of the image is cropped out. These parts are replaced by the portions of the original image I to get I''''. This completes the step 2 in the watermark recovery algorithm in Section 2. See Figure 2 for these operations.

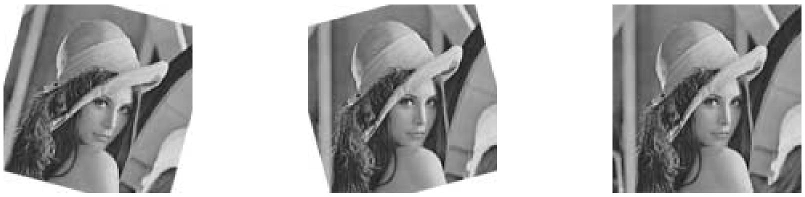


Fig. 2. Rotation of 15^o, back and glue.

Next we run the rest of the watermark recovery algorithm to get back the key k''. The key is compared with each key k' from the database. For all the keys in the database we get the similarity factors around 50% except the correct key, whose similarity factor is very high. We present that maximum similarity factor in Table 1.

Table 2 provides the results of cropping on the watermarked image from different positions. In this case also, we replace the cropped portions from the part of the original image. Table 3 provides the results of resizing. The original image size is 256×256. Table 4 provides the results related to resizing, rotation and cropping. We take the watermarked image I'. Then we consecutively go for first resizing, rotation and back, resizing back and finally glue parts from original image.

Next we present the results of intentional random attacks. First consider the results in Table 5. During the simulation we randomly change (either increase or decrease) the value of each pixel of the watermarked image by certain percentage of the original pixel value. We find that our scheme survives this attack

Table 1. Rotation.

Angle	Similarity
10^o	95.1%
20^o	92.5%
30^o	87.6%
40^o	80.1%
45^o	76.8%

Table 2. Cropping.

Cropping %	Position	Similarity
14	Vertically Left	92.1%
14	Vertically Middle	93.1%
14	Vertically Right	92.8%
14	Horizontally Below	94.8%
14	Horizontally Up	93.9%
28	Vertically Left	85.4%
28	Vertically Middle	86.2%
28	Vertically Right	84.8%
28	Horizontally Below	89.3%
28	Horizontally Up	91.4%

Table 3. Resizing.

Modified Size	Similarity
128×128	90.8%
153×153	92.1%
180×180	96.0%
204×204	97.6%
230×230	98.4%
307×307	99.1%
332×332	99.2%

Table 4. Resizing, Rotation and Cropping.

Modified Size	Rotation	Similarity
128×128	15^o	76.9%
128×128	30^o	73.4%
128×128	45^o	64.8%
192×192	15^o	88.0%
192×192	30^o	77.9%
192×192	45^o	70.0%
320×320	15^o	94.1%
320×320	30^o	90.0%
320×320	45^o	78.9%

satisfactorily. In fact for a noise of 10% in spatial domain, the watermarked image degrades to a high extent (see left of the Figure 3). Even then we can find the similarity of the order of 67.7%. In Table 6, we present the robustness of our algorithm against combined attacks in spatial domain. We first resize the watermarked image. Then we provide some amount of rotation. After that we introduce 2% noise (random increase or decrease of pixel values) in spatial domain. We then rotate back the image, resize it back to its original size and then glue the cropped portions by that of the original image.

Table 5. Noise in spatial domain.

% Change	Similarity
1%	98.6%
2%	93.9%
3%	86.7%
4%	86.7%
5%	78.2%

Table 6. Combined attack in spatial domain.

Resize	Rotation	Similarity
128 × 128	15^o	62.9%
192 × 192	15^o	83.9%
192 × 192	30^o	73.3%
192 × 192	45^o	67.4%
320 × 320	15^o	89.1%
320 × 320	30^o	84.3%
320 × 320	45^o	74.8%

In some watermarking schemes, the watermark is incorporated by modifying the least significant bit (lsb) of each pixel value [9, Page 29] and a standard attack is to modify those bits to remove the watermark. Given an watermarked image, setting each lsb to 1 (respectively 0) we get a similarity factor 99% (respectively 84%). Complementing each lsb we get the similarity factor 81%.

We now present the result after applying some filters on the watermarked image. We take the filter mask to be 3×3. For median filter we find the similarity factor 87.2%. Average filtering gives a similarity factor of 80.18%. Let us also consider the effect of low pass filtering on the watermarked image. We use 0.9 intensity value [5]. The similarity factor in this case is 79.8%.

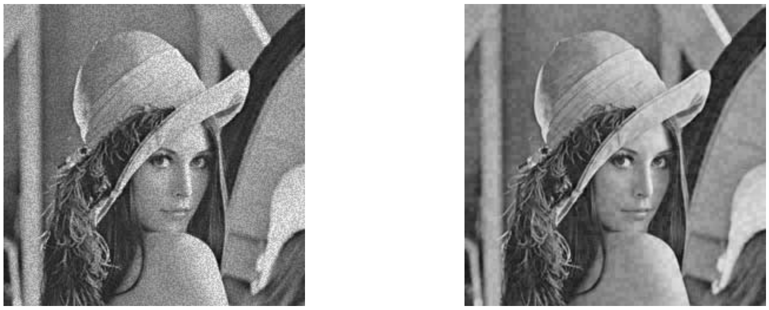


Fig. 3. Left: Noise of 10%. Right: Blocking attack with a value 5.

Since we are proposing a block oriented scheme, it is natural to consider that the attacker will try to do some block oriented cryptanalysis. The most natural attack in this direction could be as follows. The attacker selects each block in the watermarked image and then modify all the pixels of the block by some value x. As example, one block is selected and the pixels corresponding to that block are increased by 2. Similarly, another block is selected and all the pixels of that block are decreased by 2. This decision of either increasing or decreasing all the pixels is taken randomly. With this kind of attack we get the similarity values 83.4%, 64.2%, 59.6%, 56.7%, 53.6% for the block oriented change in pixel values 1, 2, 3, 4, 5 respectively. It is true that the similarity factor goes closer towards 50% for change in pixel value 5. However, in that situation the watermarked image quality degrades considerably (see right of Figure 3).

Next we consider the rewatermarking attack. Note that this is also one type of blocking attack. In this attack we rewatermark the watermarked image once again and then try to recover the original watermark. Consider the original image as I. We first apply the watermark with the keys π_1, k_1 on I to get I_1 and then rewatermark I_1 with π_2, k_2 to get I_2. As we run the watermark recovery algorithm, we get a similarity of 85.2% with the key k_1 using permutation π_1.

Note that this is only a basic attack and an attacker may go on exercising the rewatermarking attack more than once. We consider two different cases here. In the first case the attacker goes on rewatermarking for number of steps. After a few steps we find that the similarity factor with the original key decreases and then settles around 50%. However, in such a case the quality of the image degrades considerably. We represent this in graphical manner. In the horizontal axis we present the iteration number and in the vertical axis we present the similarity factor and the PSNR with respect to the original image. Note that the similarity factor comes down to 55% after 16th step and at this point the PSNR value is as low as +17.7 db with respect to the first copy of the watermarked image. Thus the image quality degrades to a large extent and such an attack can not be successfully performed.

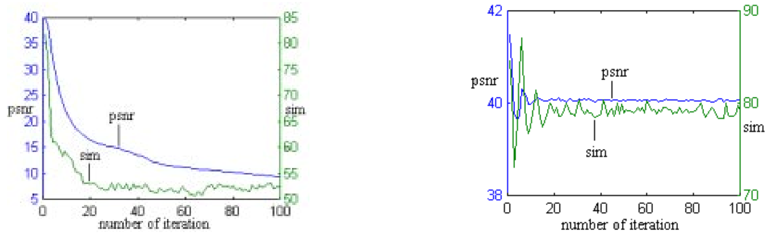


Fig. 4. Rewatermarking, left : without replacement, right : with replacement.

In such a scenario, an attacker can modify the scheme as follows. Consider that the attacker decides a limit of change in the pixel values ($-\delta$ to $+\delta$) in comparison to the watermarked image he possesses. This will keep the image quality intact. For the experiment we choose $\delta = 3$. That is, in such a case the attacker takes the watermarked image I' as reference. Let the pixel values of I' be denoted as p'. After each step of rewatermarking, the attacker checks whether the pixel value is in the limit $p' - \delta$ to $p' + \delta$. If it is not in this limit, the value is selected uniformly at random from the range $p' - \delta$ to $p' + \delta$. This keeps the quality of the image very good (PSNR around +40 db) with respect to the first copy of the original image. In such a situation, we find the similarity factor with the original key stays at a very high range, i.e., in the range 73% to 85%. Though quality of image stays good, such an attack cannot destroy the watermark.

3.2 Transformations and Attacks in Frequency Domain

We first consider the JPEG compression [16]. This compression is executed in the DCT (Discrete Cosine Transform) domain. We compress the TIF formatted watermarked image (using Electric Eye on Linux 6.1) to JPEG format giving different values of compression. Then we get back the compressed image to TIF format. Even if we use as low quality as 20% (i.e., high compression), we can recover the key with significant correlation 67.9%. At this high level of compression, the image quality degrades substantially and blocking effect is noticed. Even then it is possible to recover the watermark. This we present in Table 7. In Table 8 we consider the noise in frequency domain. We transform the image to frequency domain using FFT (Fast Fourier Transform) and then introduce some percentage of noise in terms of increasing or decreasing the values in the frequency spectrum. We proceed with inverse FFT and then use the real part of the matrix to get back the image in spatial domain.

Next we consider a combination of attacks considering both frequency and spatial domain. In Table 9 we provide results of rotation at different degrees, then introduce 5% noise in spatial domain and perform JPEG compression at quality level 50%. After that we get back the image to TIF format, rotate back the image and glue the cropped portions by the original image. In Table 10, we provide the results corresponding to the attack combining resize, rotation and noise in frequency domain. We transform the resized and rotated image by FFT.

Table 7. JPEG compression.

JPEG Quality	Similarity
20%	67.9%
30%	74.7%
50%	86.5%
60%	90.0%
70%	94.4%
80%	97.9%
99%	99.3%

Table 9. Rotation, noise, JPEG.

Rotation	Similarity
10^o	74.2%
10^o	72.6%
30^o	70.6%
40^o	66.7%

Table 8. Noise in FFT domain.

% noise	Similarity
1%	94.6%
2%	77.3%
3%	76.5%
4%	63.9%
5%	61.7%

Table 10. Resize, rotation, FFT, noise.

Resize	Rotation	Similarity
128×128	15^o	63.9%
192×192	15^o	72.4%
192×192	30^o	66.3%
320×320	15^o	81.5%
320×320	30^o	72.3%

In the frequency domain we introduce 2% random noise. Next we use the inverse transform to get back the image in spatial domain (the real part of the matrix after inverse FFT), rotate it back, resize it to original size and glue cropped portions from the original image.

Next we consider transformations and attacks in wavelet domain. Currently image processing techniques using wavelet transformation have received a lot of attention. Moreover, it is used in JPEG 2000 standard [19] for compression. We here concentrate on the transformation in wavelet domain. We consider two different strategy for attacking. In the first attack we transform the watermarked image to wavelet domain. Then we introduce random noise of 1% to 10% in the wavelet domain. Again we get back the image in spatial domain and run the watermark recovery algorithm. The results are presented in Table 11. We now present another attack which is analogous to compression in wavelet domain. The strategy is to select a range of values in the wavelet transform domain and then replace them by zero. In Table 12 we present the range of values that is replaced by zero in wavelet domain and how the similarity factor varies after inverse wavelet transform.

Table 11. Random Noise.

% noise	similarity
1%	93.5%
2%	81.7%
3%	77.1%
4%	73.2%
5%	70.8%
10%	62.8%

Table 12. Replacement by zeros.

range	similarity
-1 to +1	97.2%
-2 to +2	97.0%
-3 to +3	96.7%
-4 to +4	96.5%
-5 to +5	96.4%
-10 to +10	96.2%

3.3 Nonlinear Geometric Attack (Stirmark)

Here we present the robustness of our scheme against the famous Stirmark attacks. We use the softwares available at [18]. For the version 1 of Stirmark attack we get a similarity factor of 71.4%, whereas for version 3 of the Stirmark attack we get the similarity factor of 70.9%. We use the default parameters used in the programs available at [18]. The watermark uses a block size of 4×4 and the watermarked images after Stirmark attacks are presented in Figure 5.

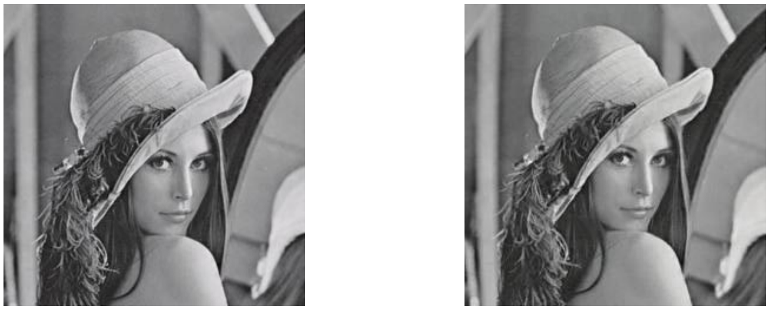


Fig. 5. Watermarked images after Stirmark 1 (left) and Stirmark 3 (right) attacks.

Much better result can be achieved by increasing the block size to 8×8 (in this case the larger window size used in the window matching algorithm is 12×12). Here, $n = 1024$ and $m = 665$. For version 1 of Stirmark we get a similarity factor of 83.4% and for stirmark version 3 the similarity factor achieved is 86.46%. Due to the increased size of β the window matching algorithm performs better and we get improved results.

Let us now highlight the utility of window matching algorithm. If we do not use this, the similarity factors for 4×4 block size are 53.5% and 55.5% for strimark version 1 and Stirmark version 3. For 8×8, the values are 62.5% and 60.5% for Stirmark version 1 and Stirmark version 3. Thus, it is clear how the window matching algorithm improves the robustness of our scheme against small nonlinear geometric transformations.

3.4 Collusion Attack

In this section we consider the scenario when two or more buyers collude together to remove the watermarking information from the image. It is now clear that the standard invisible watermarking schemes are prone from collusion attacks under a very general framework [4]. Still we perform the benchmark tests as provided in [3] and show that our scheme survives at least these benchmark attacks.

We take 3 different keys k_1, k_2, k_3 to generate 3 watermarked images I_1, I_2, I_3 from I. These three users collude in the following way. We take the 3 different pixels p_1, p_2, p_3 from the same location of the three images I_1, I_2, I_3. Then we construct a pixel value $p = f(p_1, p_2, p_3)$ where f is taken to be one of the functions from median, $\max, \min$, average and weighted average. Taking the pixel values p we construct an image J. We then consider J as attacked image and run recovery algorithm on this. We also consider a 4th key k_4 for a 4th buyer who has not participated in the collusion. The results are shown in Table 13. It is clear from the data that for the buyers who have participated in the collusion are identified, since the similarity factors are much greater than 50%. On the other hand, the buyer who has not participated in the collusion is not wrongly implicated since the similarity is very close to 50%. Note that in the weighted average, we calculate $p = \lfloor \frac{0.5p_1+0.25p_2+0.25p_3}{3} \rfloor$. It is clear that in this case the maximum correlation is found with the first buyer.

Table 13. Collusion attack.

Strategy	Similarity #1	Similarity #2	Similarity #3	Similarity #4
median	74.1%	75.6%	74.8%	50.6%
min	62.0%	61.9%	62.0%	49.6%
max	62.6%	62.5%	62.5%	50.6%
average	74.1%	75.6%	74.8%	50.4%
wtd. avg.	86.7%	62.1%	61.5%	50.1%

Similar to the collusion attack experiment proposed in [3] we check the scenario considering 5 watermarked images and taking their averages. We get the similarity of 65.6%, 66.3%, 64.5%, 66.6%, 66.3% with the participated keys. On the other hand we have taken two other watermarked images which were not used in collusion. For those two keys the similarity factors are 49.8%, 50.1%.

4 Conclusion

In this paper we have presented a simple but robust invisible digital watermarking scheme. We demonstrated the robustness of the scheme with an exhaustive set of image transformation and intentional attacks. It is also important to state that in all the experiments the scheme never provided any false positive alarm. In all the cases, the experiments with a wrong key provides a similarity factor very close to 50% and never implicates an honest buyer.

References

1. W. Bender, D. Gruhl and N. Morimoto. Techniques for Data Hiding. In *Proceedings of the SPIE 2420, Storage and Retrieval for Image and Video Databases III*, Pages 164–173, 1995.
2. I. J. Cox and M. L. Miller. A review of watermarking and the importance of perceptual modeling. In *Proceedings of Electronics Imaging*, February 1997.
3. I. J. Cox, J. Kilian, T. Leighton and T. Shamoon. Secure Spread Spectrum Watermarking for Multimedia. *IEEE Trans. on Image Proc.*, 6(12):1673–1687, 1997.
4. F. Ergun, J. Kilian and R. Kumar. A note on the limits of collusion-resistant watermarks. In *Eurocrypt 1999*, LNCS, Springer Verlag, 1999.
5. R. C. Gonzalez and P. Wintz. *Digital Image Processing.* Addison-Wesley Publishing (MA, USA), 1988.
6. F. Hartung and M. Kutter. Multimedia watermarking techniques. *Proceedings of IEEE*, 87(7), July 1999.

7. J. R. Hernansez, M. Amadeo and F. P. Gonzalez. DCT-Domain Watermarking Techniques for Still Images: Detector Performance Analysis and a New Structure. *IEEE Transactions on Image Processing*, 9(1):55–68, 2000.
8. N. F. Johnson, Z. Duric and S. Jajodia. Information Hiding: Steganography and Watermarking – Attacks and Countermeasures. Kluwer Academic Publishers Boston, USA, 2000.
9. S. Katzenbeisser, F. A. P. Petitcolas. Information Hiding Techniques for Steganography and Digital Watermarking. Artech House, USA, 2000.
10. G. C. Langelaar and R. L. Lagendijk. Optimal Differential Energy Watermarking of DCT Encoded Images and Video. *IEEE Transactions on Image Processing*, 10(1):148–158, 2001.
11. C. S. Lu, S. K. Huang, C. J. Sze and H. Y. Liao. *A New Watermarking Technique for Multimedia Protection*. Multimedia Image and Video Processing. L. Guan, S. Y. Kung and J. Larsen (edited), pages 507–530, CRC Press, 2001.
12. F. J. MacWillams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. North Holland, 1977.
13. F. A. P. Petitcolas, R. J. Anderson, M. G. Kuhn and D. Aucsmith. Attacks on Copyright Marking Systems. In *2nd Workshop on Information Hiding*, pages 218–238 in volume 1525 of Lecture Notes in Computer Science. Springer Verlag, 1998.
14. F. A. P. Petitcolas and R. J. Anderson. Evaluation of Copyright Marking Systems. In *IEEE Multimedia Systems*, Florence, Italy, June 1999.
15. J. O. Ruanaidh, H. Petersen, A. Herrigel, S. Pereira and T. Pun. Cryptographic copyright protection for digital images based on watermarking techniques. *Theoretical Computer Science* 226:117–142, 1999.
16. G. K. Wallace. The JPEG still picture compression standard. *Comm. of the ACM*, April 1991.
17. M. M. Yeung. Digital Watermarking. *Comm. of the ACM*, 41(7), July 1998.
18. http://www.cl.cam.ac.uk/˜ fapp2/watermarking/stirmark/
19. http://www.jpeg.org/JPEG2000.htm

A Flexibly Revocable Key-Distribution Scheme for Efficient Black-Box Tracing

Tatsuyuki Matsushita

TOSHIBA Corporate Research & Development Center
1, Komukai Toshiba-cho, Saiwai-ku, Kawasaki 212-8582, Japan
`tatsuyuki.matsushita@toshiba.co.jp`
`Tel: +81-44-549-2156 Fax: +81-44-520-1841`

Abstract. We propose a new type of revocation scheme for efficient public-key black-box traitor tracing. Our revocation scheme is flexible in the sense that any number of subscribers can be revoked in each distribution under an assumption that the number of revoked subscribers who collude in one coalition is limited to a threshold, while the maximum number of revoked ones cannot be changed in previous schemes. The flexibility in revocation is significant since flexible revocation can be integrated with efficient black-box tracing and this integration can be achieved without a substantial increase in the transmission overhead over the previous schemes. In this paper, we present an efficient public-key revocable and black-box-traceable scheme by combining flexible revocation with two known black-box-tracing algorithms.

Keywords: Flexible revocation, Traitor tracing, Black-box tracing

1 Introduction

Consider the following content-distribution system: A data supplier distributes digital contents (e.g., a movie) to subscribers over a broadcast channel. Since the contents should be available only to subscribers, the data supplier broadcasts the encrypted contents. Only subscribers can decrypt them with their corresponding decryption keys given in advance. This system can be applied to pay-TV, CD(DVD)-ROM distribution, online databases, etc. In this content-distribution system, malicious subscribers may redistribute their decryption keys to non-subscribers. This piracy should be prevented since it infringes the copyright of the digital contents.

As a deterrent to the piracy, *traitor tracing* has been studied extensively. The concept of traitor tracing was introduced by Chor-Fiat-Naor [4]. In traitor tracing, each subscriber is given a distinct decryption key (*personal key*) which is contained in a decryption device (*decoder*), and the data supplier broadcasts both the contents encrypted with a *session key* and the encrypted session key (*header*). Typically, a symmetric key is used as the session key. Subscribers can obtain the session key (and consequently the contents) by inputting the received header to their decoders. In this scenario, malicious subscribers (*traitors*) may give away their personal keys to a pirated version of a decoder (*pirate decoder*). Once the pirate decoder is found, at least one of the traitors who join the piracy can be identified by it. A traitor-tracing scheme discourages traitors from committing the piracy since the confiscated pirate decoder can be traced back to its producers.

Traitor-tracing schemes can be classified into two types according to their constructions: The construction of one type of scheme is combinatorial while that of the other is algebraic and number-theoretic. The former type of scheme [4,7,9] is inefficient in the following criteria: each subscriber's storage and the transmission overhead. This is because it has to greatly degrade the efficiency in order to eliminate the probability that an honest subscriber is falsely detected as a traitor. On the other hand, an algebraic and number-theoretic approach solves the above efficiency problem. The latter type of scheme [5,3] is efficient in the sense that each subscriber's storage is constant (e.g., 160 bits) and the transmission overhead is linear only in the maximum number of traitors in a coalition. Our interest is in this type of scheme.

Among efficient public-key traitor-tracing schemes, the schemes of [8,12,10,11] achieve revocation of subscribers by using a key-distribution method proposed in [1]. In these schemes, the data supplier can revoke a certain number of subscribers in each distribution, i.e., the data supplier can make their decoders useless without confiscating them. The schemes are suitable for the content-distribution system where the subscribers frequently cancel/renew their subscriptions to the contents. Moreover, the schemes have another desirable property. The revocation mechanism can also be used to support *black-box tracing*, which provides the assurance that traitors can be identified no matter how the pirate decoder is implemented. In black-box tracing, a tracer does not break open the pirate decoder but uses it as a black box. Briefly, the tracer chooses suspects and tests whether traitors are among them only by observing the behavior of the pirate decoder on chosen inputs. If revocation is supported, the tracer can perform the above test by revoking the suspects in the input. Therefore, the revocation mechanism can seamlessly be integrated with black-box tracing.

Unfortunately, in the previous schemes [8,12,10,11], the revocation mechanism cannot be integrated with *efficient* black-box tracing, i.e., the black-box-tracing algorithm of the previous schemes has to be exponential, hence impractical, in order to keep the transmission overhead efficient. This trade-off greatly spoils the scalability of the schemes. In this paper, we get rid of the trade-off between the transmission overhead and the running time of the tracing algorithm by combining the scheme of [1] and that of [5]. Our scheme is the first one which satisfies all of the following properties:

Flexible revocation. While the number of revoked subscribers in the previous schemes is limited to a certain threshold which cannot be changed unless the system is initialized again, in our scheme, the upper bound on this parameter can be set differently in each distribution without any reinitialization. Note that even though there is a threshold of the coalition size of traitors, revocation of any number of subscribers can be achieved in the following case we consider in the paper: A set of revoked subscribers includes one or more disjoint subsets, $\mathcal{X}_1, \ldots, \mathcal{X}_i$. Each $\mathcal{X}_j$ is a distinct coalition in which revoked ones collude and $|\mathcal{X}_j|$ is limited to the threshold. Our main contribution is providing a revocation mechanism which can drastically reduce the running time of the black-box-tracing algorithm without a substantial increase in the transmission overhead over the previous schemes. Thanks to flexible revocation, our scheme can achieve the efficient transmission overhead and efficient running time of the tracing algorithm, while the previous ones cannot accomplish both.

Efficient black-box tracing. For completeness of our scheme, we give an explicit description of two black-box-tracing algorithms, although they are mentioned in [12]. The first can identify all of the traitors in a coalition with running time $O(n)$. The second can detect at least one of them with running time $O(\log n)$, while the running time of the previous tracing algorithm is $O(\binom{n}{k})$, where n is the total number of subscribers and k is the maximum coalition size. The tracing algorithms can work under the same threat model as assumed in the previous schemes. Besides, the tracing result is publicly verifiable in our scheme since no secret information is needed to execute the tracing algorithms, i.e., anyone can work as a tracer. This property provides a stronger deterrent to the piracy. Moreover, it is useful in the case where subscribers are given their decoders in which the personal keys have already been embedded. Since a malicious merchant might deceive subscribers into buying pirate decoders as legitimate ones, they want to be convinced that decoders to be purchased are genuine without breaking open their decoders. Black-box tracing with public verifiability enables subscribers to verify the legitimacy of their decoders by executing a tracing algorithm with their decoders as an input.

Public-key setting. In our scheme, anyone can work as a data supplier since the session key is encrypted with the public key.

The rest of the paper is organized as follows. In Sect. 2, an assumption on the pirate decoder is described. We propose an efficient public-key revocable and black-box-traceable scheme in Sect. 3. The proposed scheme is analyzed in terms of security and efficiency in Sect. 4 and Sect. 5, respectively. We present our conclusions in Sect. 6.

2 Model of Pirate Decoders

We adopt the same assumption on the pirate decoder as in the previous schemes.

Assumption 1 The pirate decoder constructed by traitors always outputs the correct plaintext if it gets the header of a regular form, i.e., it does not take measures that might fool the tracer.

It is possible for traitors to construct a pirate decoder that can take measures that fool the tracer whatever it receives as the header. However, such a decoder could hardly be an item for sale. For example, consider that the digital data of a movie is distributed. The data is divided into 3-minute lengths, and each part of the data is encrypted with a different session key. Traitors can build the pirate decoder which intentionally fails to calculate the session key every few minutes to escape from the analysis, but no one want to buy such a decoder because only fragments of the movie can be watched by using it. Therefore, Assumption 1 is not strong.

3 Proposed Scheme

First, we describe an outline of the proposed scheme. Secondly, the construction of our scheme is shown.

3.1 Outline

Our scheme consists of the four phases.
Initialization: A trusted party generates and secretly gives subscribers their personal keys. The personal key is stored in the decoder.
Encryption: The data supplier selects revoked subscribers and encrypts (i) the digital contents with the session key and (ii) the session key itself as the header in such a way that revoked ones cannot obtain the session key in the header. Then, the data supplier broadcasts the encrypted digital contents and the header. To avoid complication, we assume that (i) the symmetric encryption algorithm, which is used for encryption of the contents, is secure and publicly known and (ii) a broadcast channel is reliable in the sense that the received information is not altered.
Decryption: When receiving the header, subscribers input it to their decoders in order to obtain the session key. Since only non-revoked subscribers can compute the session key, the digital contents are available only to them.
Tracing: Suppose that the pirate decoder is confiscated. The tracer chooses a set of suspects and builds the header in which the selected suspects are revoked. The tracer inputs the chosen headers to the pirate decoder and observes whether it decrypts the headers correctly or not. If its output is (i) correct on the input where a set of revoked suspects is $\mathcal{T}$ and (ii) incorrect on the input where a set of revoked suspects is $\mathcal{T} \cup \{u\}$, then the tracer decides the subscriber, u, is a traitor.

3.2 Protocol

Let n be the total number of subscribers and k be the maximum number of traitors in a coalition. Let p, q be primes s.t. $q|p-1$ and $q \geq n+k+1$. Let g be a qth root of unity over $\mathbb{Z}_p^*$ and G_q be a subgroup of $\mathbb{Z}_p^*$ of order q. Let $\mathcal{U}$ be a set of subscribers ($\mathcal{U} \subseteq \mathbb{Z}_q \backslash \{0\}$) and $\mathcal{X}$ be a set of revoked subscribers. All the participants agree on p, q, and g. The calculations are done over $\mathbb{Z}_p^*$ unless otherwise specified.
Initialization: Choose $a_0, \ldots, a_k, b_1, \ldots, b_k \in_{\mathrm{R}} \mathbb{Z}_q$. Then, compute the public key e as follows:

$$\begin{aligned} e &= (g, y_{0,0}, \ldots, y_{0,k}, y_{1,1}, \ldots, y_{1,k}) \\ &= (g, g^{a_0}, \ldots, g^{a_k}, g^{b_1}, \ldots, g^{b_k}). \end{aligned}$$

Split $\mathcal{U}$ into k disjoint subsets $\mathcal{U}_1, \ldots, \mathcal{U}_k$. These subsets are publicly known. Suppose that $u \in \mathcal{U}_i$. The subscriber u's personal key is $(u, f_i(u))$ where

$$\begin{aligned} f_i(x) &= \sum_{j=0}^{k} a_{i,j} x^j \bmod q, \\ a_{i,j} &= \begin{cases} a_j & (i \neq j), \\ b_j & (i = j). \end{cases} \end{aligned}$$

Encryption: Check whether $\mathcal{Y} \overset{\triangle}{=} \mathcal{X} \setminus \cup_{j \in \{z | \mathcal{U}_z \subseteq \mathcal{X}\}} \mathcal{U}_j$ is a non-empty set or not. If $\mathcal{Y} = \{x_1, \ldots, x_w\}$, then find an integer d s.t. $d(k+1) \leq w \leq d(k+1) + k$. Set $m \leftarrow d(k+1) + k$. Otherwise ($\mathcal{Y} = \emptyset$ or $\mathcal{X} = \emptyset$), set $m \leftarrow k$ and $w \leftarrow 0$.

Select $c_0, \ldots, c_m \in_{\mathrm{R}} \mathbb{Z}_q$, and $x_{w+1}, \ldots, x_m \in_{\mathrm{R}} \mathbb{Z}_q \setminus (\mathcal{U} \cup \{0\})$ if $w < m$. Then, build the header $h(r, \mathcal{X})$ as follows:

$$h(r, \mathcal{X}) = (h, h_{0,0}, \ldots, h_{0,m}, h_{1,1}, \ldots, h_{1,m}, H_1, \ldots, H_m),$$

where

$$\begin{aligned} h &= g^r, \\ h_{0,j} &= y_{0,z_j}^r g^{c_j}, \\ z_j &= j \bmod (k+1), \\ h_{1,j} &= \begin{cases} g^{r_j} & (\mathcal{U}_{z_j} \subseteq \mathcal{X},\ z_j \neq 0) \\ y_{1,z_j}^r g^{c_j} & (\mathcal{U}_{z_j} \not\subseteq \mathcal{X},\ z_j \neq 0) \end{cases} \\ H_j &= (x_j, g^{F(x_j)}), \\ F(x) &= \sum_{j=0}^{m} c_j x^j \bmod q, \end{aligned}$$

and r, r_j for all $j \in \{z \mid 1 \leq z \leq m,\ z \not\equiv 0 \pmod{(k+1)},\ \mathcal{U}_{z \bmod (k+1)} \subseteq \mathcal{X}\}$ are random numbers generated by the data supplier. If $z_j = 0$, then $h_{1,j}$ is not included in the header. Each 2-tuple H_j is a distinct share of $g^{F(0)} = g^{c_0}$, which is the session key.

Decryption: Suppose that $x_0 \in \mathcal{U}_i$. If $x_0 \notin \mathcal{X}$, the subscriber, x_0, can correctly compute one share of $g^{F(0)}$, i.e., $(x_0, g^{F(x_0)})$ as follows:

$$g^{F(x_0)} = D_i(x_0) \Big/ h^{f_i(x_0) \sum_{j=0}^{d} x_0^{j(k+1)}},$$

where $d = (m-k)/(k+1)$ and

$$\begin{aligned} D_i(x_0) &= \prod_{j=0}^{m} B_{i,j}^{x_0^j}, \\ B_{i,j} &= \begin{cases} h_{0,j} & (i \neq j \bmod (k+1)), \\ h_{1,j} & (i = j \bmod (k+1)). \end{cases} \end{aligned}$$

Since $\mathcal{U}_1, \ldots, \mathcal{U}_k$ are publicized, the subscriber, x_0, knows a value of i s.t. $x_0 \in \mathcal{U}_i$. Therefore, $D_i(x_0)$ can be calculated as follows:

$$\begin{aligned} D_i(x_0) &= \prod_{j=0}^{d} \left(h_{0,j(k+1)} \times h_{0,j(k+1)+1}^{x_0} \times \cdots \times h_{1,j(k+1)+i}^{x_0^i} \times \cdots \times h_{0,j(k+1)+k}^{x_0^k} \right)^{x_0^{j(k+1)}} \\ &= \prod_{\ell=0}^{d} \left(g^{r \sum_{j=0}^{k} a_{i,j} x_0^j} \right)^{x_0^{\ell(k+1)}} \times g^{\sum_{j=0}^{m} c_j x_0^j} \\ &= h^{f_i(x_0) \sum_{\ell=0}^{d} x_0^{\ell(k+1)}} \times g^{F(x_0)}. \end{aligned}$$

Now, the subscriber, x_0, obtains the $m+1$ shares $H_1, \ldots, H_m$, and $(x_0, g^{F(x_0)})$. Then, compute the session key, $g^{F(0)}$, by performing the Lagrange interpolation in the exponents:

$$\begin{aligned} g^{F(0)} &= \prod_{j=0}^{m} \left(g^{F(x_j)} \right)^{L_j} \\ &= g^{\sum_{j=0}^{m} L_j F(x_j)}, \end{aligned}$$

where

$$L_j = \prod_{0 \le \ell \le m, \ell \ne j} \frac{x_\ell}{x_\ell - x_j} \bmod q.$$

Tracing: We describe how two efficient black-box-tracing algorithms are applied to our scheme.

Algorithm 1 (One-by-one black-box tracing)
Input: $\mathcal{U}_1, \ldots, \mathcal{U}_k$, and the pirate decoder, $\mathcal{D}$.
Output: a set of traitors, $\mathcal{R}$.

Step 1. Label all the elements in $\mathcal{U}_1, \ldots, \mathcal{U}_k$ as follows:

$$\begin{aligned} \mathcal{U}_1 &= \{u_1, \ldots, u_{d_1}\}, \\ \mathcal{U}_2 &= \{u_{d_1+1}, \ldots, u_{d_1+d_2}\}, \\ &\vdots \\ \mathcal{U}_k &= \{u_{\sum_{j=1}^{k-1} d_j + 1}, \ldots, u_{\sum_{j=1}^{k} d_j}\}. \end{aligned}$$

Recall that $\cup_{i=1}^{k} \mathcal{U}_i = \mathcal{U}$ and $\mathcal{U}_i \cap \mathcal{U}_j = \emptyset$ if $i \ne j$, and therefore $\sum_{j=1}^{k} d_j = n$.

Step 2. Set $\mathcal{R} \leftarrow \emptyset$ and $\ell \leftarrow 1$. Then, repeat the following procedures for $1 \le \ell \le n$:

(2-1) Set $\mathcal{T}_\ell \leftarrow \mathcal{U} \backslash \{u_\ell\}$ and execute the algorithm $\mathcal{A}_{\text{test}}$ with input $\mathcal{U}_1, \ldots,$ $\mathcal{U}_k$, $\mathcal{T}_\ell, \mathcal{D}$. The algorithm $\mathcal{A}_{\text{test}}$ is described below.

(2-2) If the output of $\mathcal{A}_{\text{test}}(\mathcal{U}_1, \ldots, \mathcal{U}_k, \mathcal{T}_\ell, \mathcal{D})$ is "correct", then set $\mathcal{R} \leftarrow \mathcal{R} \cup \{u_\ell\}$.

(2-3) Increment ℓ by one and go to (2-1).

Step 3. Output $\mathcal{R}$ as a set of traitors.

Algorithm 2 (Black-box test $\mathcal{A}_{\text{test}}$)
Input: $\mathcal{U}_1, \ldots, \mathcal{U}_k$, $\mathcal{T}_\ell$, and the pirate decoder, $\mathcal{D}$.
Output: "correct" or "incorrect".

Step 1. It is obvious that for some $i \in \{1, \ldots, k\}$ there exists at most one set $\mathcal{U}_i$ s.t. $\mathcal{U}_i \not\subseteq \mathcal{T}_\ell$ and $\mathcal{U}_i \cap \mathcal{T}_\ell \neq \emptyset$. Check whether there exists such a set or not.
- If there is such a set $\mathcal{U}_i$, then set $\mathcal{Y} \leftarrow \mathcal{U}_i \cap \mathcal{T}_\ell$.
- Otherwise (there is no such set), set $\mathcal{Y} \leftarrow \emptyset$.

Then, build $h(r, \mathcal{T}_\ell)$ in the same way as in Sect. 3.

Step 2. Input $h(r, \mathcal{T}_\ell)$ to $\mathcal{D}$ and observe its output.
- If $\mathcal{D}$ outputs the correct session key, $g^{F(0)}$, on the input, then output "correct".
- Otherwise ($\mathcal{D}$ outputs incorrectly), output "incorrect".

In the ℓth test of Algorithm 1, the tracer examines whether the subscriber, u_ℓ, is a traitor or not. Since all the subscribers are checked one by one, all the traitors who give away their personal keys to the pirate decoder can be identified. The next tracing algorithm can more efficiently detect at least one of the traitors by using binary search.

Algorithm 3 (Binary-search black-box tracing)
Input: $\mathcal{U}_1, \ldots, \mathcal{U}_k$, and the pirate decoder, $\mathcal{D}$.
Output: a traitor's ID.

Step 1. Relabel $\mathcal{U}_1, \ldots, \mathcal{U}_k$ as $\mathcal{V}_1, \ldots, \mathcal{V}_k$ in randomized order. For $1 \le i \le k$, label all the elements of $\mathcal{V}_i$ in randomized order. Then denote $\mathcal{V}_1, \ldots, \mathcal{V}_k$ as follows:

$$\begin{aligned} \mathcal{V}_1 &= \{u_1, \ldots, u_{d'_1}\}, \\ \mathcal{V}_2 &= \{u_{d'_1+1}, \ldots, u_{d'_1+d'_2}\}, \\ &\vdots \\ \mathcal{V}_k &= \{u_{\sum_{j=1}^{k-1} d'_j+1}, \ldots, u_{\sum_{j=1}^{k} d'_j}\}. \end{aligned}$$

Note that $\sum_{j=1}^{k} d'_j = n$, $\mathcal{V}_i \in \{\mathcal{U}_1, \ldots, \mathcal{U}_k\}$, and $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$ if $i \neq j$.

Step 2. Set $Lo \leftarrow 0$, $Hi \leftarrow n$, and $\ell \leftarrow 1$. For $1 \le \ell \le \lfloor \log_2 n \rfloor + 1$, repeat the following procedures:

(2-1) Set $Mid \leftarrow \lfloor (Lo+Hi)/2 \rfloor$ and $\mathcal{T}_\ell \leftarrow \{u_1, \ldots, u_{Mid}\}$. Then, execute the algorithm $\mathcal{A}_{\text{test}}$ (described in Algorithm 2) with input $\mathcal{U}_1, \ldots, \mathcal{U}_k, \mathcal{T}_\ell, \mathcal{D}$.
- If $\mathcal{A}_{\text{test}}(\mathcal{U}_1, \ldots, \mathcal{U}_k, \mathcal{T}_\ell, \mathcal{D}) =$"correct", then set $Lo \leftarrow Mid$.
- Otherwise (the output is "incorrect"), set $Hi \leftarrow Mid$.

(2-2) Increment ℓ by one and go to (2-1).

Step 3. After the $\lfloor \log_2 n \rfloor + 1$ tests, there exists one element $u \in \mathcal{U}$ which satisfies the following two conditions for some $\ell \in \{1, \ldots, \lfloor \log_2 n \rfloor + 1\}$:

$$\begin{aligned} \mathcal{A}_{\text{test}}(\mathcal{U}_1, \ldots, \mathcal{U}_k, \mathcal{T}_\ell, \mathcal{D}) &= \text{"correct"}, \\ \mathcal{A}_{\text{test}}(\mathcal{U}_1, \ldots, \mathcal{U}_k, \mathcal{T}_\ell \cup \{u\}, \mathcal{D}) &= \text{"incorrect"}. \end{aligned}$$

Output u as a traitor's ID.

The same procedure as in Step 1 of Algorithm 1 can be substituted for the permutation procedure in Step 1 of Algorithm 3 if it suffices to detect only one of the traitors who are responsible for the pirate decoder, although the tracer can identify more than one of them by repeating Algorithm 3 with differently permutated $\mathcal{U}_1, \ldots, \mathcal{U}_k$.

4 Security

The security of our scheme is based on the difficulty of the Decision Diffie-Hellman problem (DDH) [2]. Informally, the assumption that DDH in G_q is intractable means that no probabilistic polynomial-time (p.p.t. for short) algorithm can distinguish with non-negligible advantage between the two distributions $\langle g_1, g_2, g_1^a, g_2^a \rangle$ and $\langle g_1, g_2, g_1^a, g_2^b \rangle$ where $g_1,\ g_2 \in_{\mathrm{R}} G_q$ and $a, b \in_{\mathrm{R}} \mathbb{Z}_q$.

4.1 Secrecy

Recall that $\mathcal{X}$ is a set of revoked subscribers and a set of subscribers $\mathcal{U}$ is divided into k disjoint subsets $\mathcal{U}_1, \ldots, \mathcal{U}_k$.

Lemma 1 *Suppose that the subscribers in $\mathcal{X}$ are revoked after obtaining a certain number (bounded by a polynomial) of previous session keys and headers, the new header, the public key, and their personal keys. For any $\mathcal{X}$, the computational complexity for any coalition of k revoked subscribers to distinguish the session key corresponding to the new header from a random element in G_q is as difficult as DDH in G_q.*

The proof is omitted due to space limitation. Lemma 1 leads to the next theorem which shows that, for any $\mathcal{X}$, no coalition of at most k revoked subscribers can compute the session key with non-negligible probability.

Theorem 1 *Suppose that the subscribers in $\mathcal{X}$ are revoked after obtaining a certain number (bounded by a polynomial) of previous session keys and headers, the new header, the public key, and their personal keys. For any $\mathcal{X}$, the computational complexity for any coalition of at most k revoked subscribers to compute the session key corresponding to the new header is at least as difficult as DDH in G_q.*

Proof Let $\mathcal{C}$ be a set of revoked subscribers in a coalition. Let $\mathcal{A}_{\mathcal{C},\mathcal{X}}^{\mathrm{comp}}$ be a p.p.t. algorithm the coalition $\mathcal{C}$ uses to compute the session key corresponding to the new header when a set of revoked subscribers is $\mathcal{X}$. Let $\mathcal{A}_{\mathcal{C},\mathcal{X}}^{\mathrm{dist}}$ be a p.p.t. algorithm the coalition $\mathcal{C}$ uses to distinguish the session key corresponding to the new header from a random element in G_q when a set of revoked subscribers is $\mathcal{X}$. Let $\mathcal{A}^{\mathrm{DDH}}$ be a p.p.t. algorithm which solves DDH in G_q. For two p.p.t. algorithms $\mathcal{A}_0, \mathcal{A}_1$, we mean by $\mathcal{A}_0 \Rightarrow \mathcal{A}_1$ that the existence of $\mathcal{A}_0$ implies that of $\mathcal{A}_1$ and by $\mathcal{A}_0 \Leftrightarrow \mathcal{A}_1$ that $\mathcal{A}_0 \Rightarrow \mathcal{A}_1$ and $\mathcal{A}_1 \Rightarrow \mathcal{A}_0$. Among $\mathcal{A}_{\mathcal{C},\mathcal{X}}^{\mathrm{comp}}, \mathcal{A}_{\mathcal{C},\mathcal{X}}^{\mathrm{dist}}$, and $\mathcal{A}^{\mathrm{DDH}}$ there exist the following three relations:

(R1) $\mathcal{A}^{\text{comp}}_{\mathcal{C},\mathcal{X}} \Rightarrow \mathcal{A}^{\text{dist}}_{\mathcal{C},\mathcal{X}}$ for any $\mathcal{X}$, $\mathcal{C}$ with $\mathcal{C} \subseteq \mathcal{X}$, $|\mathcal{C}| = k$ ($\because$ $\mathcal{A}^{\text{dist}}_{\mathcal{C},\mathcal{X}}$ can be constructed via one oracle call to $\mathcal{A}^{\text{comp}}_{\mathcal{C},\mathcal{X}}$.)

(R2) $\mathcal{A}^{\text{dist}}_{\mathcal{C},\mathcal{X}} \Leftrightarrow \mathcal{A}^{\text{DDH}}$ for any $\mathcal{X}$, $\mathcal{C}$ with $\mathcal{C} \subseteq \mathcal{X}$, $|\mathcal{C}| = k$ ($\because$ Lemma 1)

(R3) $\mathcal{A}^{\text{comp}}_{\mathcal{C}',\mathcal{X}} \Rightarrow \mathcal{A}^{\text{comp}}_{\mathcal{C},\mathcal{X}}$ for any $\mathcal{X}$, $\mathcal{C}$, and $\mathcal{C}'$ with $\mathcal{C} \subseteq \mathcal{X}$, $|\mathcal{C}| = k$, $\mathcal{C}' \subsetneq \mathcal{C}$

It follows from (R1) and (R2) that $\mathcal{A}^{\text{comp}}_{\mathcal{C},\mathcal{X}} \Rightarrow \mathcal{A}^{\text{DDH}}$ for any $\mathcal{X}$, $\mathcal{C}$ with $\mathcal{C} \subseteq \mathcal{X}$, $|\mathcal{C}| = k$. From this result and (R3), it holds that $\mathcal{A}^{\text{comp}}_{\mathcal{C},\mathcal{X}} \Rightarrow \mathcal{A}^{\text{DDH}}$ for any $\mathcal{X}$, $\mathcal{C}$ with $\mathcal{C} \subseteq \mathcal{X}$, $|\mathcal{C}| \leq k$, which means the statement of Theorem 1. □

Note that we do not consider the collusion between revoked subscribers and non-revoked ones.

4.2 Black-Box Traceability

Let $\mathcal{C}$ be a coalition of traitors and recall that $\mathcal{T}_\ell$ is a set of suspects in the input for black-box tracing. Similarly to the previous schemes, the inputs for the normal broadcast and the ones for black-box tracing are indistinguishable. Therefore, according to Assumption 1, the output of the pirate decoder on the input $h(r, \mathcal{T}_\ell)$ should be:

- the correct session key if $\mathcal{C} \nsubseteq \mathcal{T}_\ell$.
- some incorrect one with overwhelming probability if $\mathcal{C} \subseteq \mathcal{T}_\ell$ ($\because$ Theorem 1).

Lemma 2 *If Algorithm 2 answers "correct" for $\mathcal{T}_\ell$ and "incorrect" for $\mathcal{T}_\ell \cup \{u\}$ ($u \notin \mathcal{T}_\ell$), then it holds that $u \in \mathcal{C}$ with probability $1-\varepsilon$ where ε is negligible.*

Proof Since Algorithm 2 outputs "incorrect" for $\mathcal{T}_\ell \cup \{u\}$, it must hold that $\mathcal{C} \subseteq \mathcal{T}_\ell \cup \{u\}$ with overwhelming probability. Assume that $u \notin \mathcal{C}$. Since $\mathcal{C} \subseteq \mathcal{T}_\ell$ in this case, Algorithm 2 must output "incorrect" for $\mathcal{T}_\ell$ with overwhelming probability. This is a contradiction. Therefore, the case where $u \notin \mathcal{C}$ is impossible. □

In Algorithm 1, the tracer tests whether the only non-revoked subscriber, u_ℓ ($\{u_\ell\} = \mathcal{U} \backslash \mathcal{T}_\ell$), is a traitor or not. The subscriber, u_ℓ, is determined as a traitor if Algorithm 2 answers "correct" for $\mathcal{T}_\ell$.

Theorem 2 *From the pirate decoder constructed by a coalition of at most k traitors, Algorithm 1 can identify all of them with probability $1 - \varepsilon$ where ε is negligible.*

Proof According to Assumption 1, Algorithm 2 outputs "correct" for $\mathcal{T}_\ell$ if $u_\ell \in \mathcal{C}$. From Lemma 2 and the fact that $\mathcal{T}_\ell \cup \{u_\ell\} = \mathcal{U}$, it follows that $u_\ell \in \mathcal{C}$ with overwhelming probability if Algorithm 2 answers "correct" for $\mathcal{T}_\ell$. Therefore, Algorithm 2 can correctly decide whether $u_\ell \in \mathcal{C}$ with overwhelming probability. By repeating the test of Algorithm 2 for $1 \leq \ell \leq n$, Algorithm 1 can identify all the traitors with overwhelming probability. □

In Algorithm 3, the tracer finds the last traitor who is added to a set of suspects.

Theorem 3 *From the pirate decoder constructed by a coalition of at most k traitors, Algorithm 3 can identify at least one of them with probability* $1 - \varepsilon$ *where ε is negligible.*

Proof First, we prove that Algorithm 3 does not fail. Since Algorithm 2 must answer "incorrect" for $\mathcal{U}$ with overwhelming probability and "correct" for $\emptyset$, there exist with overwhelming probability two disjoint sets $\mathcal{T}, \{u\}$ ($\mathcal{T} \subsetneq \mathcal{U}, \{u\} \subseteq \mathcal{U} \backslash \mathcal{T}$) s.t. Algorithm 2 outputs "correct" for $\mathcal{T}$ and "incorrect" for $\mathcal{T} \cup \{u\}$. It is clear that Algorithm 3 can find such sets $\mathcal{T}, \{u\}$. Therefore, Algorithm 3 does not fail.

Secondly, it immediately follows from Lemma 2 that the subscriber, u, who is detected by Algorithm 3, is a traitor with overwhelming probability. This completes the proof. □

5 Efficiency

Let $\mathcal{P}$, $\mathcal{S}$, and $\mathcal{H}$ be sets of possible personal keys, session keys, and headers, respectively. In Table 1, the previous schemes and ours are compared from the viewpoints of each subscriber's storage, the transmission overhead, the running time of the tracing algorithm, and the type of revocation. In our scheme, each subscriber's storage is constant and the transmission overhead does not increase substantially over the previous schemes. For example, it holds that $\log|\mathcal{H}| = (4k+2)\log|\mathcal{S}|$ when the number of revoked subscribers is k. Note that the transmission overhead does not always grow as the number of revoked subscribers increases. In our scheme, the data supplier can make it impossible for the revoked subscribers to compute the session key by (i) adding their shares of the session key to the header or (ii) substituting a random value for the element used only by the subscribers in one of the k disjoint subsets if all of them in the subset are revoked. Therefore, if all of the members in a subset are revoked, the corresponding shares of the session key are unnecessary in the header. For instance, the transmission overhead is independent of the total number of revoked subscribers and it still holds that $\log|\mathcal{H}| = (4k+2)\log|\mathcal{S}|$ as long as $d = 0$, i.e., there is at most k revoked subscribers who coexist with one or more non-revoked ones in each of their subsets. Revocation in our scheme is flexible in the sense that the maximum number of revoked subscribers does not have to be fixed in the initialization phase and is variable in each distribution, although the security level of revocation remains unchanged. It is desirable that the capacity level of revocation should be changeable in each distribution in the case where the range of the number of revoked subscribers is wide.

The outstanding advantage of our scheme is the efficient running time of the tracing algorithm with the efficient transmission overhead. In our scheme, (i) all the traitors in a coalition can be identified with running time $O(n)$ and (ii) at least one of them can be caught with running time $O(\log n)$, while the running time of the previous one is $O(\binom{n}{k})$. Thanks to flexible revocation, an efficient tracing algorithm can be achieved without a substantial increase in the transmission overhead. Note that, as shown in Table 2, the scheme of [12] has to

Table 1. Comparison of each subscriber's storage, the transmission overhead, the running time of the tracing algorithm, and a feature (n: the total number of subscribers, k: the maximum number of traitors in a coalition, $t_{\max}$: the maximum number of revoked subscribers, d: an integer s.t. $|\mathcal{Y}| \leq d(k+1)+k$ where $\mathcal{Y}$ is a set of revoked subscribers who coexist with one or more non-revoked ones in each of their subsets)

	Each subscriber's storage and the transmission overhead	The running time of the tracing algorithm	Is $t_{\max}$ changeable in each distribution?
[8]	$\lvert\mathcal{P}\rvert = \lvert\mathcal{S}\rvert,\ \log\lvert\mathcal{H}\rvert = (2k+1)\log\lvert\mathcal{S}\rvert$	$O(\binom{n}{k})$	No ($t_{\max} = k$)
[10]	$\lvert\mathcal{P}\rvert = \lvert\mathcal{S}\rvert,\ \log\lvert\mathcal{H}\rvert = 4k\log\lvert\mathcal{S}\rvert$	$O(\binom{n}{k})$	No ($t_{\max} = 2k-1$)
[11]	$\log\lvert\mathcal{P}\rvert = 2\log\lvert\mathcal{S}\rvert,\ \log\lvert\mathcal{H}\rvert = 3(k+1)\log\lvert\mathcal{S}\rvert$	$O(\binom{n}{k})$	No ($t_{\max} = k$)
[12]	$\lvert\mathcal{P}\rvert = \lvert\mathcal{S}\rvert,\ \log\lvert\mathcal{H}\rvert = 2(k+1)\log\lvert\mathcal{S}\rvert$	$O(\binom{n}{k})$	No ($t_{\max} = k$)
Ours	$\lvert\mathcal{P}\rvert = \lvert\mathcal{S}\rvert$, $\log\lvert\mathcal{H}\rvert = \{4(d+1)k+3d+2\}\log\lvert\mathcal{S}\rvert$	$O(n)$ (Alg. 1) $O(\log n)$ (Alg. 3)	Yes

Table 2. Comparison of each subscriber's storage and the transmission overhead required to identify all the traitors in a coalition efficiently (The same notations are used as in Table 1.)

	Each subscriber's storage and the transmission overhead	The running time of the tracing algorithm
[12]	$\lvert\mathcal{P}\rvert = \lvert\mathcal{S}\rvert,\ \log\lvert\mathcal{H}\rvert = 2n\log\lvert\mathcal{S}\rvert$	$O(n)$
Ours	$\lvert\mathcal{P}\rvert = \lvert\mathcal{S}\rvert,\ \log\lvert\mathcal{H}\rvert = \{4(d+1)k+3d+2\}\log\lvert\mathcal{S}\rvert$	$O(n)$

incur an inefficient transmission overhead in order to support efficient black-box tracing, due to lack of the flexibility in revocation. This problem is common to the other previous schemes.

6 Conclusions

In this paper, we have proposed a flexible-revocation scheme for efficient public-key black-box traitor tracing. Our scheme is efficient in all of the following criteria: each subscriber's storage, the transmission overhead, and the running time of the tracing algorithm. Thanks to flexible revocation, which makes it possible to change the maximum number of revoked subscribers in each distribution, two efficient tracing algorithms can be applied to our scheme without a substantial increase in the transmission overhead. One of the two tracing algorithms can identify all of the traitors in a coalition with running time $O(n)$, and the other can detect at least one of them with running time $O(\log n)$, where n is the total number of subscribers.

As a concluding remark, we mention our future research. If more intelligent pirate decoders are taken into account, the assumptions adopted in this paper do not always hold true. For example, the camouflage attack [6], in which the pirate decoder conceals the correct output and pretends not to be able to output correctly based on some strategy, might defeat the proposed scheme. It is an open problem to construct an efficient public-key revocable and black-box-traceable scheme which satisfies a property that the information on the identities of suspects is not revealed in the inputs for black-box tracing.

References

1. J. Anzai, N. Matsuzaki, and T. Matsumoto: "A Quick Group Key Distribution Scheme with "Entity Revocation"", In *Proc. of ASIACRYPT '99*, LNCS 1716, Springer-Verlag, pp. 333–347, 1999.
2. D. Boneh: "The Decision Diffie-Hellman Problem", In *Proc. of the Third Algorithmic Number Theory Symposium*, LNCS 1423, Springer-Verlag, pp. 48–63, 1998.
3. D. Boneh and M. Franklin: "An Efficient Public Key Traitor Tracing Scheme", In *Proc. of CRYPTO '99*, LNCS 1666, Springer-Verlag, pp. 338–353, 1999.
4. B. Chor, A. Fiat, and M. Naor: "Tracing Traitors", In *Proc. of CRYPTO '94*, LNCS 839, Springer-Verlag, pp. 257–270, 1994.
5. K. Kurosawa and Y. Desmedt: "Optimum Traitor Tracing and Asymmetric Schemes", In *Proc. of EUROCRYPT '98*, LNCS 1403, Springer-Verlag, pp. 145–157, 1998.
6. T. Matsushita, K. Kobara, and H. Imai: "Revocable Black-box Tracing against Self-Defensive Pirate Decoders", In *Proc. of the 2nd Workshop on Information Security Applications (WISA '01)*, pp. 335–353, September 2001.
7. M. Naor and B. Pinkas: "Threshold Traitor Tracing", In *Proc. of CRYPTO '98*, LNCS 1462, Springer-Verlag, pp. 502–517, 1998.
8. M. Naor and B. Pinkas: "Efficient Trace and Revoke Schemes", In *Proc. of Financial Cryptography '00*, LNCS 1962, Springer-Verlag, pp. 1–20, February 2000.
9. D. R. Stinson and R. Wei: "Combinatorial Properties and Constructions of Traceability Schemes and Frameproof Codes", *SIAM Journal on Discrete Mathematics*, Vol. 11, No. 1, pp. 41–53, 1998.
10. W. Tzeng and Z. Tzeng: "A Public-Key Traitor Tracing Scheme with Revocation Using Dynamic Shares", In *Proc. of PKC '01*, LNCS 1992, Springer-Verlag, pp. 207–224, 2001.
11. Y. Watanabe, G. Hanaoka, and H. Imai: "Efficient Asymmetric Public-Key Traitor Tracing without Trusted Agents", *CT-RSA '01*, pp. 392–407, 2001.
12. M. Yoshida and T. Fujiwara: "A Subscriber-Excluding and Traitor-Tracing Broadcast Distribution System", *IEICE Trans. Fundamentals*, Vol. E84-A, No. 1, pp. 247–255, January 2001.

Low Complexity Bit Serial Systolic Multipliers over $GF(2^m)$ for Three Classes of Finite Fields

Soonhak Kwon

Institute of Basic Science and Dept. of Math., Sungkyunkwan University
Suwon 440-746, Korea
shkwon@math.skku.ac.kr

Abstract. By using a standard polynomial basis, we present a low complexity bit serial systolic multiplier over $GF(2^m)$ when there exist the following types of irreducible polynomials, $x^m + x^{m-1} + 1$, $x^m + \sum_{i=0}^{m-2} x^i$ and $\sum_{i=0}^{m} x^i$, an all one polynomial. When compared with most of other bit serial systolic multipliers, our multiplier needs two latches fewer in each basic cell. Therefore, the hardware complexity of our systolic array is approximately 20 percent reduced from other existing multipliers.

Keywords: finite field, basis, systolic multiplier, all one polynomial

1 Introduction

Arithmetic of finite fields, especially finite field multiplication, found various applications in many cryptographic areas. Therefore an efficient design of a finite field multiplier is needed. Though one may design a finite field multiplier in a software arrangement, a hardware implementation has a strong advantage when one wants a high speed multiplier. Moreover, arithmetic of $GF(2^m)$ is easily realized in a circuitry using a few logical gates. A good multiplication algorithm depends on the choice of a basis for a given finite field. In general, there are three types of basis being used, that is, polynomial, dual and normal basis. Some popular multipliers for cryptographic and coding theoretical purposes are Berlekamp's bit serial multipliers [1,2] which use a dual basis, and bit parallel multipliers of Massey-Omura type [3,4,5] which use a normal basis. Above mentioned multipliers and other traditional multipliers have some unappealing characteristics. For example, they have irregular circuit designs. In other words, their hardware structures may be quite different for varying choices of m for $GF(2^m)$, though the multiplication algorithm is basically same for each m. This is a great drawback when one has to use two finite fields where one field is a subfield of the other, which arises in many cryptographic situations. Moreover as m gets large, the propagation delay also increases. So deterioration of the performance is inevitable. A systolic multiplier does not suffer from above problems. It has a regular structure consisting of a number of replicated basic cells, each of which has the same circuit design. So overall structures of systolic multipliers are same and not depending on a particular choice of m for $GF(2^m)$. Furthermore since each basic cell is only connected with its neighboring cells, signals can be propagated at a high clock speed. Accordingly, the computational delay of the non systolic multipliers in [1,2,3,4,5] is very long when compared with systolic multipliers if m is large. There are systolic multipliers using a polynomial basis [7,8] and a dual basis [9,10,11]. When one uses a polynomial basis to multiply two elements in a finite field, there are basically two types of multiplication algorithms, namely, LSB (least significant bit) first scheme and MSB (most significant bit) first scheme. To find a bit serial systolic arrangement, MSB first scheme is used by Wang and Lin in [7] and LSB first scheme is used by Yeh et al. in [8]. A design in [8] has a better longest path delay than that of [7] due to increased parallelism among internal computations. However one needs two control signals in [8] whereas [7] needs only one. Therefore the hardware complexity of [8] is higher than that of [7] because of the extra latches (flip-flops). In this paper, we modify MSB first scheme used in [7] so that we eliminate two latches in each basic cell when there exists an irreducible all one polynomial of degree m for $GF(2^m)$. We show that this multiplier has the best hardware complexity among all currently known bit serial systolic multipliers. Also we propose a low complexity multiplier for each irreducible polynomial, $x^m + x^{m-1} + 1$ and $x^m + \sum_{i=0}^{m-2} x^i$. We compare our multipliers with other existing multipliers and show that the number of necessary latches in each basic cell is reduced by two from other bit serial systolic multipliers.

2 MSB First Algorithm

Let $GF(2^m)$ be a finite field of 2^m elements. $GF(2^m)$ is a vector space over $GF(2)$ of dimension m. We briefly explain basic finite field arithmetic and MSB first scheme. Let $F(x) = x^m + f_{m-1}x^{m-1} + \cdots + f_1 x + f_0 \in GF(2)[x]$ be an irreducible polynomial over $GF(2)$. Then we have $GF(2^m) = GF(2)[x]/(F(x))$ and an element of $GF(2^m)$ is uniquely represented by a polynomial of degree less than m, $A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0 \in GF(2)[x]$. Let $B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1 x + b_0$ be another element in $GF(2^m)$. A multiplication of two elements $A(x)$ and $B(x)$ is given by

$$P(x) = A(x)B(x) \pmod{F(x)}.$$

Let $T_0(x) = 0$ and for each $1 \le i \le m$, define

$$T_i(x) = A(x)(b_{m-1}x^{i-1} + b_{m-2}x^{i-2} + \cdots + b_{m-i}) \pmod{F(x)}.$$

Then we have $P(x) = T_m(x)$ and

$$\begin{aligned} T_i(x) &= A(x)(b_{m-1}x^{i-1} + b_{m-2}x^{i-2} + \cdots + b_{m-i}) \pmod{F(x)} \\ &= xA(x)(b_{m-1}x^{i-2} + b_{m-2}x^{i-3} + \cdots + b_{m-i+1}) \pmod{F(x)} + A(x)b_{m-i} \\ &= xT_{i-1}(x) \pmod{F(x)} + A(x)b_{m-i}. \end{aligned}$$

Letting $T_i(x) = t_{i,1}x^{m-1} + t_{i,2}x^{m-2} + \cdots + t_{i,m-1}x + t_{i,m}$, we have the recursive relation

$$T_i(x) = xT_{i-1}(x) + t_{i-1,1}F(x) + b_{m-i}A(x), \quad 1 \le i \le m.$$

Comparing the coefficients of x^{m-k} of above polynomials, we get

$$t_{i,k} = t_{i-1,k+1} + t_{i-1,1}f_{m-k} + b_{m-i}a_{m-k}, \quad 1 \le i, k \le m.$$

Wang and Lin [7] realized above algorithm in the following bit serial systolic arrangement. For convenience we assume $m = 4$. Fig. 1(a) is the circuit of ith basic cell and Fig. 1(b) is the corresponding systolic array. The multiplier supports a pipelined operation with latency $3m$ and throughput rate $1/m$.

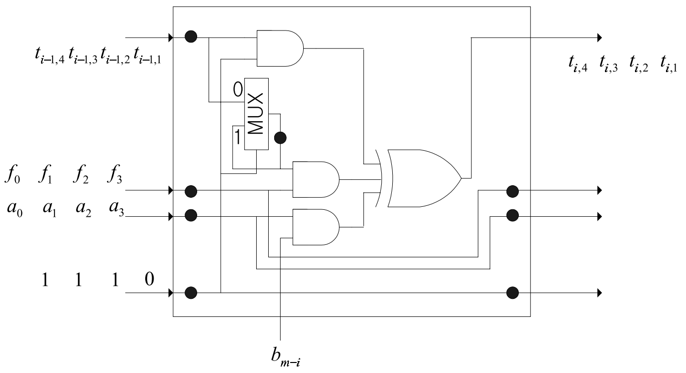

$t_{i-1,4}$ $t_{i-1,3}$ $t_{i-1,2}$ $t_{i-1,1}$
$t_{i,4}$ $t_{i,3}$ $t_{i,2}$ $t_{i,1}$
0
MUX
1
f_0 f_1 f_2 f_3
a_0 a_1 a_2 a_3
1 1 1 0
b_{m-i}

Fig. 1(a). The circuit of ith basic cell in $GF(2^4)$

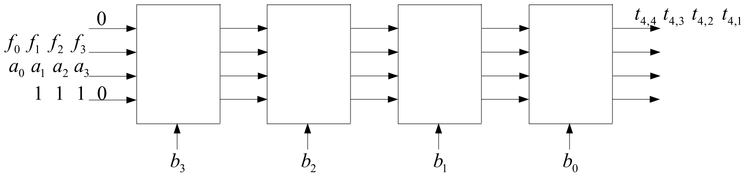


Fig. 1(b). Corresponding systolic array

3 Multipliers of Low Complexity Using Special Types of Irreducible Polynomials

3.1 A Multiplier Using an Irreducible All One Polynomial

A polynomial of the form $1 + x + x^2 + \cdots + x^m$ in $GF(2)[x]$ is called an all one polynomial (AOP) of degree m over $GF(2)$. It is well known [5,6] that an AOP of degree m is irreducible over $GF(2)$ if and only if $m + 1 = p$ is a prime and 2 is a primitive root modulo p. For example, an AOP of degree

$m = 4$ is irreducible over $GF(2)$. When the irreducible polynomial $F(x) = f_0 + f_1x + \cdots + f_{m-1}x^{m-1} + x^m \in GF(2)[x]$ used in the multiplier in Fig. 1 is AOP, i.e. $f_i = 1$ for all $0 \leq i \leq m-1$, we may further reduce the hardware complexity of the systolic design. This is easily noticed when one sees that the inputs f_i are not necessary because $t_{i-1,1}f_{m-k} = t_{i-1,1}$, $1 \leq k \leq m$, in the recursive relation stated above Fig. 1(a). Therefore one can eliminate two latches and one AND gate in Fig. 1(a), and the resulting circuit is shown in Fig. 2 for the case $m = 4$.

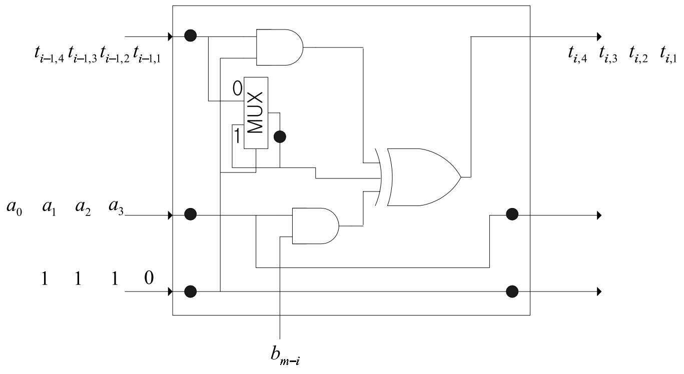

$t_{i-1,4}$ $t_{i-1,3}$ $t_{i-1,2}$ $t_{i-1,1}$
$t_{i,4}$ $t_{i,3}$ $t_{i,2}$ $t_{i,1}$
0
MUX
1
a_0 a_1 a_2 a_3
1 1 1 0
b_{m-i}

Fig. 2(a). The circuit of ith basic cell using irreducible AOP

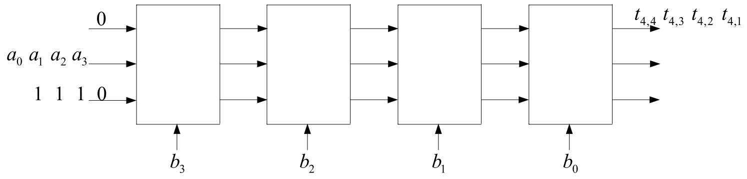


Fig. 2(b). Corresponding systolic array in $GF(2^4)$ where $F(x) = 1 + x + x^2 + x^3 + x^4$

An irreducible all one polynomial (AOP) is closely related with a normal basis of some good properties. In fact, if α is a zero of an irreducible AOP of degree m, then $\{\alpha, \alpha^2, \alpha^3, \cdots, \alpha^m\}$ is known [6] to be a normal basis in $GF(2^m)$ and is widely used [3,5,6] in the design of a bit parallel multiplier of low complexity. In some literature [6], this normal basis is called an optimal normal basis of type I. However our basis in the systolic design in Fig. 2 is a polynomial basis, $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$. To the author's knowledge, this is the first design of a bit serial systolic multiplier using an irreducible AOP or an optimal normal basis of type I, though there are some other examples [10,11] of bit serial systolic multipliers of low complexity using irreducible trinomials of a special kind, namely, $x^m + x + 1$. As one can see from Table 1, our design of an AOP multiplier needs fewer MUXs and gates when compared with the ones using $x^m + x + 1$. A multiplier using irreducible $x^m + x + 1$ is presented in [10] as a special case of the construction of a dual basis bit serial systolic multiplier. A similar construction (again using a dual basis with the same polynomial) is also proposed in [11].

Table 1. Comparison of cell complexity

	basis with $F(x)$	AND	XOR	3XOR	MUX	Latch	longest path delay
[7]	polynomial	3	0	1	1	8	$D_A+D_{3X}+D_L$
[8]	polynomial	3	2	0	1	10	$D_A+D_X+D_L$
[9]	dual	2	2	0	2	8	$D_A+D_X+D_M+D_L$
[10]	dual x^m+x+1	4	3	0	1	6	$D_A+2D_X+D_L$
[11]	dual x^m+x+1	3 + one NAND	2	0	2	6	$D_A+D_X+D_N+D_L$
Fig. 2	polynomial AOP	2	0	1	1	6	$D_A+D_{3X}+D_L$
Fig. 3	polynomial $x^m+x^{m-1}+1$	3 + one NAND	0	1	1	6	$D_A+D_{3X}+D_N+D_L$
Fig. 4	polynomial $x^m+\sum_{i=0}^{m-2}$	3	0	1	1	6	$D_A+D_{3X}+D_L$

All above multipliers have latency m and throughput rate $1/m$. AND, NAND, XOR and MUX mean 2-input gates and 3XOR means a 3-input XOR gate. D_A, D_N, D_X and D_M mean the delay time of an AND gate, a NAND gate, a XOR gate and a MUX respectively. Note that the area complexity of latch is higher than any other gates in the table. Approximately, a latch takes 5 times more area than an AND gate, 2.5 times more than a MUX and 2 times more than a XOR gate.

Since these multipliers do not need the inputs of f_i, one can save two latches from a usual multiplier.

3.2 A Multiplier Using an Irreducible $x^m + x^{m-1} + 1$

An irreducible polynomial $F(x) = x^m + x^{m-1} + 1 \in GF(2)[x]$ corresponds to input signal $\langle f_0\ f_1\ f_2\ \cdots\ f_{m-1}\rangle = \langle 1\ 0\ \cdots\ 0\ 1\rangle$. By adjusting control signal $\langle 1\ 1\ \cdots\ 1\ 0\rangle$, we can make the signal $\langle f_0\ f_1\ f_2\ \cdots\ f_{m-1}\rangle$ occur without actually using the inputs f_i. That is, by taking a NAND gate of control signal $\langle 1\ 1\ \cdots\ 1\ 0\rangle$ and one clock delayed control signal $\langle 1\ \cdots\ 1\ 0\ 1\rangle$, we have the signal $\langle 0\ \cdots\ 0\ 1\ 1\rangle$. We make this signal come in the cell one clock ahead of the signals a_i. Therefore the resulting signal is $\langle 1\ 0\ \cdots\ 0\ 1\rangle$, which is the input signal of $x^m + x^{m-1} + 1$. The circuit is shown in Fig. 3 for the case $m = 4$. We omit the corresponding systolic array since it is exactly same to Fig. 2(b).

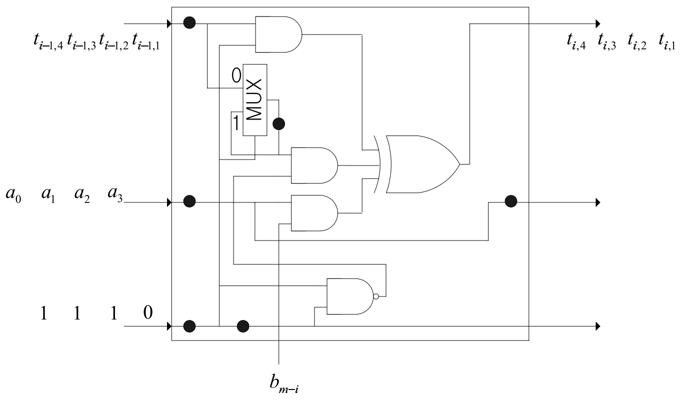

$t_{i-1,4}\ t_{i-1,3}\ t_{i-1,2}\ t_{i-1,1}$
$t_{i,4}\ t_{i,3}\ t_{i,2}\ t_{i,1}$
0
1
MUX
$a_0\ a_1\ a_2\ a_3$
1 1 1 0
b_{m-i}

Fig. 3. The circuit of basic cell using irreducible $x^m + x^{m-1} + 1$

It is easy to show that the polynomial $x^m + x^{m-1} + 1$ is irreducible over $GF(2)$ if and only if $x^m + x + 1$ is irreducible, since $1/\alpha$ is a root of $x^m + x + 1$ whenever α is a root of $x^m + x^{m-1} + 1$. Therefore, the dual basis multipliers in [10,11] using $x^m + x + 1$ and our multiplier using $x^m + x^{m-1} + 1$ are applicable for the same class of finite fields. Again, Table 1 says that our systolic arrangement in Fig. 3 has a lower hardware complexity than the multipliers in [10,11]. It should be mentioned that the polynomial $x^m + x + 1$ (or $x^m + x^{m-1} + 1$) is not very often irreducible over $GF(2)$ when compared with AOP. It is known [14] that there are 21 of $m \leq 1000$ for which $x^m + x + 1$ (or $x^m + x^{m-1} + 1$) is irreducible over $GF(2)$. However the number of $m \leq 1000$ for which an irreducible AOP of degree m exists [6,p.100] is 68. Therefore an AOP multiplier is applicable to a broader class of m than a multiplier using $x^m + x + 1$.

3.3 A Multiplier Using an Irreducible $x^m + \sum_{i=0}^{m-2} x^i$

It is also possible that we do not use any gate on the control signals and still we can get an efficient multiplier. For example, when $F(x) = x^m + \sum_{i=0}^{m-2} x^i$, the corresponding input $\langle f_0\ f_1\ f_2\ \cdots\ f_{m-1}\rangle$ is same to the control signal $\langle 1\ 1\ \cdots\ 1\ 0\rangle$. Thus we have a multiplier for an irreducible $F(x) = x^m + \sum_{i=0}^{m-2} x^i$ and it is shown in Fig. 4, where the case $m = 5$ is dealt since $x^5 + x^3 + x^2 + x + 1$ is irreducible over $GF(2)$.

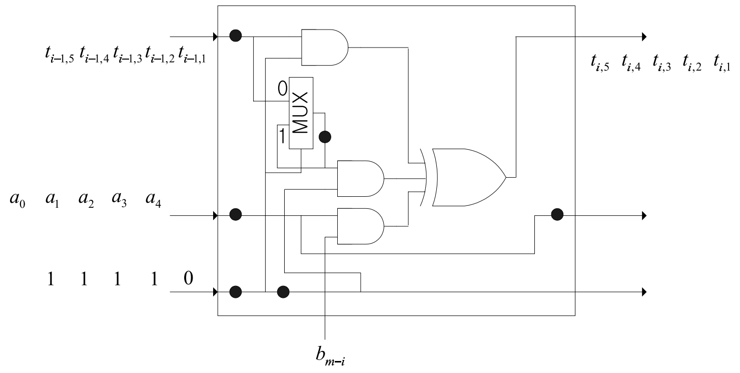

$t_{i-1,5}$ $t_{i-1,4}$ $t_{i-1,3}$ $t_{i-1,2}$ $t_{i-1,1}$
$t_{i,5}$ $t_{i,4}$ $t_{i,3}$ $t_{i,2}$ $t_{i,1}$
0
1
MUX
a_0 a_1 a_2 a_3 a_4
1 1 1 1 0
b_{m-i}

Fig. 4. The circuit of basic cell using irreducible $F(x) = x^m + \sum_{i=0}^{m-2} x^i$

4 Conclusions

In this paper, we proposed three types of low complexity bit serial systolic multipliers applicable for some classes of finite fields. A slight modification of the MSB first design in [7] yields efficient multipliers applicable to the following types of irreducible polynomials, namely, an AOP polynomial, $x^m + x^{m-1} + 1$ and $x^m + \sum_{i=0}^{m-2} x^i$. We showed that all these multipliers need only 3 (serial) input lines into the basic cells as shown in Table 1. Therefore, they have lower hardware complexities than other multipliers with 4 (serial) input lines. Moreover, since we use a standard polynomial basis, we do not have to worry about the basis conversion process. The irreducibility of an AOP polynomial and $x^m + x^{m-1} + 1$ is well understood for moderately small values of m. It is known [14] that the values of $m \leq 1000$ for which $x^m + x^{m-1} + 1$ is irreducible over $GF(2)$ are $m = 2, 3, 4, 6, 7, 9, 15, 22, 28, 30, 46, 60, 63, 127, 153, 172, 303, 471, 532, 865, 900$. There are 21 of them. The number of $m \leq 2000$ for which an AOP of degree m is irreducible [6,p.100] is 118. For example, we have an irreducible AOP of degree m when $m = 2, 4, 10, 12, 18, 28, 36, 52, 58, 60, 66, 82, 100, 106, 130, 138, 148, 162, 172, 178, 180, \cdots$. For a cryptographic purpose where one needs a large finite field $GF(2^m)$, one can always find a sequence of m for which $GF(2^m)$ has an AOP basis or a trinomial basis of $x^m + x^{m-1} + 1$. The irreducibility of $x^m + \sum_{i=0}^{m-2} x^i$ is not currently available in a known literature. But it is not difficult to test the irreducibility of $x^m + \sum_{i=0}^{m-2} x^i$ by using standard algorithms explained in [6]. By a hand calculation, one easily shows that $x^m + \sum_{i=0}^{m-2} x^i$ is irreducible when $m = 3, 5, 7$ but reducible when $m = 9, 11$. It would be very nice if one has a table of the irreducibility of this polynomial and compares the result with those of an AOP polynomial and $x^m + x^{m-1} + 1$, which will clarify possible finite fields for which our multipliers have applications.

References

1. E.R. Berlekamp, "Bit-serial Reed-Solomon encoders," *IEEE Trans. Inform. Theory*, vol. 28, pp. 869–874, 1982.
2. M. Wang and I.F. Blake, "Bit serial multiplication in finite fields," *SIAM J. Disc. Math.*, vol. 3, pp. 140–148, 1990.
3. M.A. Hasan, M.Z. Wang and V.K. Bhargava, "A modified Massey-Omura parallel multiplier for a class of finite fields," *IEEE Trans. Computers*, vol. 42, pp. 1278–1280, 1993.
4. B. Sunar and Ç.K. Koç, "An efficient optimal normal basis type II multiplier," *IEEE Trans. Computers*, vol 50, pp. 83–87, 2001.
5. T. Itoh and S. Tsujii, "Structure of parallel multipliers for a class of finite fields $GF(2^m)$," *Information and computation*, vol. 83, pp. 21–40, 1989.
6. A.J. Menezes, *Applications of finite fields*, Kluwer Academic Publisher, 1993.
7. C.L. Wang and J.L. Lin, "Systolic array implementation of multipliers for finite fields $GF(2^m)$," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 796–800, 1991.
8. C.S. Yeh, I.S. Reed and T.K. Troung, "Systolic multipliers for finite fields $GF(2^m)$," *IEEE Trans. Computers*, vol. C-33, pp. 357–360, 1984.
9. S.T.J. Fenn, M. Benaissa and D. Taylor, "Dual basis systolic multipliers for $GF(2^m)$," *IEE Proc. Comput. Digit. Tech.*, vol. 144, pp. 43–46, 1997.
10. J.J. Wozniak, "Systolic dual basis serial multiplier", *IEE Proc. Comput. Digit. Tech.*, vol. 145, pp. 237–241, 1998.
11. M. Diab and A. Poli, "New bit serial systolic multiplier for $GF(2^m)$ using irreducible trinomials," *Electronics Letters*, vol. 27, pp. 1183–1184, 1991.
12. S.K. Jain, L. Song and K.K. Parhi, "Efficient semisystolic architectures for finite field arithmetic," *IEEE Trans. VLSI Syst.*, vol. 6, pp. 101–113, 1998.
13. C. Paar, P. Fleischmann and P. Roelse, "Efficient multiplier archtectures for Galois fields $GF(2^m)$," *IEEE Trans. Computers*, vol. 47, pp. 162–170, 1998.

14. D.R. Stinson, "On bit serial multiplication and dual bases in $GF(2^m)$," *IEEE Trans. Inform. Theory*, vol. 37, pp. 1733–1736, 1991.
15. C.Y. Lee, E.H. Lu and J.Y. Lee, "Bit parallel systolic multipliers for $GF(2^m)$ fields defined by all one and equally spaced polynomials," *IEEE Trans. Computers*, vol. 50, pp. 385–393, 2001.
16. C.W. Wei, "A systolic power sum circuit for $GF(2^m)$," *IEEE Trans. Computers*, vol. 43, pp. 226–229, 1994.

Fast Elliptic Curve Multiplications with SIMD Operations

Tetsuya Izu[1] and Tsuyoshi Takagi[2]

[1] FUJITSU LABORATORIES Ltd.
4-1-1, Kamikodanaka, Nakahara-ku, Kawasaki, 211-8588, Japan
izu@flab.fujitsu.co.jp
[2] Technische Universität Darmstadt, Fachbereich Informatik
Alexanderstr.10, D-64283 Darmstadt, Germany
ttakagi@cdc.informatik.tu-darmstadt.de

Abstract. The Single Instruction, Multiple Data (SIMD) architecture enables to compute in parallel on a single processor. The SIMD operations are implemented on some processors such as Pentium 3/4, Athlon, SPARC, and even on smart cards. This paper proposes efficient algorithms for assembling an elliptic curve addition (ECADD), doubling (ECDBL), and k-iterated ECDBL (k-ECDBL) with SIMD operations. Using the singed binary chain, we can compute a scalar multiplication about 10% faster than the previously fastest algorithm by Aoki et al. Combined with the sliding window method or the width-w NAF window method, we also achieve about 10% faster parallelized scalar multiplication algorithms with SIMD operations. For the implementation on smart cards, we propose two fast parallelized scalar multiplication algorithms with SIMD resistant against side channel attacks.

Keywords: Elliptic Curve Cryptosystems (ECC), scalar multiplication, NAF, window method, SIMD operations, side channel attacks

1 Introduction

Elliptic curve cryptosystems (ECC) have become a vital technology for cryptography because of their high security with shorter key-length and faster computation than existing other cryptographic schemes. Designers are strongly recommended to implement ECC based on the standards [IEEE,NIST,SEC]. Let $E(K)$ be an elliptic curve over a finite field $K = \mathbb{F}_q$ (q is a power of a prime p). The dominant computation of all ECC algorithms, including the encryption/decryption and the signature generation/verification, is the *scalar multiplication* $d * P$ for a point $P \in E(K)$ and an integer d. Numerous algorithms have been proposed to enhance the computing time of the scalar multiplication (see [BHLM01]). In this paper we deal with the parallel computation of the scalar multiplication. Parallelized computation of elliptic curve arithmetic is discussed by Koyama and Tsuruoka for the first time [KT92]. In the paper, they assumed that the parallel computation is executed by a special hardware. Nowadays SIMD (Single instruction, Multiple Data) operations, with which multiple data are processed by a single instruction on a single processor, are available on major CPUs; MMX and SSE for Pentium 3/4, enhanced 3D Now! for Athlon, VIS for SPARC, and even on smart cards. In this context, Smart proposed a family of elliptic curves suitable for the SIMD operations [Sma01], however these curves are not fully-compatible to standard elliptic curves. Aoki et al. proposed efficient algorithms for computing addition formulas in parallel with SIMD [AHKO01]. They implicitly assumed the time for computing multiplication by a constant over the definition field is negligible, and their addition formulas are not suitable for arbitrary computational environment. Their computation time for computing ECADD and ECDBL are $5M + 1S$ and $2M + 3S$, where M and S are the computation time of a multiplication and a squaring of the definition field, respectively.

In this paper, we propose complete and efficient algorithms for computing scalar multiplications in parallel with SIMD for standard curves. We show the faster implementation of an elliptic curve addition (ECADD) and doubling (ECDBL) in the Jacobian coordinate as well as k-iterated ECDBL (k-ECDBL). The algorithms of our proposed addition formulas are written using only basic operations of the definition field, namely multiplications, squarings, additions, and subtractions. We also optimize the number of auxiliary variables required for the formulas. Thus the addition formulas can be used for all computation environments. Let A be the computation time of an addition or a subtraction of the definition field. The proposed ECADD requires time $4M + 2S + 6A$ with 8 auxiliary variables. It is faster than the formula by Aoki et al. over the environment whose squaring is faster. The proposed ECDBL requires $2M+3S+7A$ with 7 auxiliary variables. The formula k-ECDBL aims at computing k-time ECDBLs with one formula instead of repeatedly applying ECDBL. For non-parallel cases, Itoh et al. proposed an efficient algorithm for the k-iterated ECDBL in the Jacobian coordinate [ITTTK99]. We develop the SIMD version of their algorithm. The computation time of our k-`ECDBL` with SIMD requires $2kM+(2k+1)S+7kA$, and we can reduce the number of squarings for $k > 1$. We then apply these formulas for computing scalar multiplications. Both the signed binary method and the window-based methods are discussed. Through the paper we assume that the base point is not fixed. The signed binary method using our proposed formulas can yield about 10% faster scalar multiplication than one using the formulas by Aoki et al. We deal with two window-based schemes, namely the sliding window method (Algorithm 14.85 of [MvOV97]) and the width-w NAF window method (Algorithm 11 of [BHLM01]). Because these methods have long zero runs of their addition chain, we can expect that the algorithm k-`ECDBL` effectively reduces the number of squaring. The improvement of our schemes over the previous schemes are estimated about 10%. We made experiments for the expected length of the zero run, and we confirmed our above estimations are correct.

The shorter key-length of ECC is suitable for implementing on smart cards. However, the side channel attacks (SCA) are serious if there is a tight connection between a secret key in the device and side information such as computing times and power consumptions [Koc96,KJJ99]. There are two levels of SCA at the moment; the simple power analysis (SPA) and the differential power analysis (DPA). Implementer should pay attention against each of them. Recently, Fischer et al. proposed a parallelized scalar multiplication algorithm resistant against SCA on the CRYPTO2000 [FGKS02]. The CRYPTO2000 is an arithmetic co-processor on which SIMD operations are available. In this paper, we propose two parallelized scalar multiplication algorithms resistant against SCA. The first one is a direct application of proposed ECADD and ECDBL. The second one is based on the Möller's window-based algorithm [Möl01]. Comparing with previous algorithms, we establish the fastest algorithm for computing parallelized scalar multiplication with SIMD operations.

2 Preliminaries

Elliptic Curves: In this paper we assume $K = \mathbb{F}_p$ $(p > 3)$ be a finite field with p elements. Elliptic curves over K can be represented by the Weierstrass form equation

$$E(K) := \{(x, y) \in K \times K \mid y^2 = x^3 + ax + b \; (a, b \in K,\ 4a^3 + 27b^2 \neq 0)\} \cup \mathcal{O}, \quad (1)$$

where $\mathcal{O}$ is the point of infinity. An elliptic curve $E(K)$ has an additive group structure. For two points $P_1,\ P_2$ of $E(K)$, we call $P_1 + P_2$ $(P_1 \neq P_2)$ the elliptic curve addition (ECADD) and $P_1 + P_2$ $(P_1 = P_2)$, that is $2 * P_1$, the elliptic curve doubling (ECDBL). Let d be an integer and P be a point on the elliptic curve $E(K)$. The scalar multiplication is to compute the point $d * P$.

Jacobian Coordinate System: Costs of computing ECADD/ECDBL depend on the representation of an elliptic curve [CMO98]. The *Jacobian coordinate* offers a faster computation, which is obtained by setting $x = X/Z^2,\ y = Y/Z^3$ in (1). The equation of the curve is given by $E_J : Y^2 = X^3 + aXZ^4 + bZ^6$ and a point on the curve is represented by (X, Y, Z) where two points (X, Y, Z) and $(\lambda^2 X, \lambda^3 Y, \lambda Z)$ $(\lambda \neq 0)$ are identified as same. The addition formulas for Jacobian coordinate are given in Table 1.

Table 1. Addition formulas in Jacobian coordinate

ECADD	ECDBL
Input: $P_1 = (X_1, Y_1, Z_1), P_2 = (X_2, Y_2, Z_2)$	Input: $P_1 = (X_1, Y_1, Z_1),\ a$
Output: $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$	Output: $P_4 = 2 * P_1 = (X_4, Y_4, Z_4)$
$U_1 \leftarrow X_1 Z_2^2,\ U_2 \leftarrow X_2 Z_1^2$	$M \leftarrow 3X_1^2 + aZ_1^4$
$S_1 \leftarrow Y_1 Z_2^2,\ S_2 \leftarrow Y_2 Z_1^3$	$S \leftarrow 4X_1 Y_1^2$
$H \leftarrow U_2 - U_1,\ r \leftarrow S_2 - S_1$	$T \leftarrow -2S + M^2$
$X_3 \leftarrow -H^3 - 2U_1 H^2 + r^2$	$X_4 \leftarrow T$
$Y_3 \leftarrow -S_1 H^3 + r(U_1 H^2 - X_3)$	$Y_4 \leftarrow -8Y_1^4 + M(S - T)$
$Z_3 \leftarrow Z_1 Z_2 H$	$Z_4 \leftarrow 2Y_1 Z_1$

Computing times for ECADD and ECDBL in Table 1 are `ECADD` $= 12M + 4S$ and `ECDBL` $= 4M + 6S$ [CMO98], where M, S denotes computing times of a multiplication and a squaring in K, respectively. If one of the Z-coordinate equals 1, `ECADD` is reduced to $\mathtt{ECADD}_{Z=1} = 8M + 3S$. If $a = -3$, `ECDBL` is reduced to $\mathtt{ECDBL}_{a=-3} = 4M + 4S$.

Addition Chain: Let d be an n-bit integer and P be a point of the elliptic curve E. A standard way for computing a scalar multiplication $d*P$ is to use the binary expression $d = d[n-1]2^{n-1} + d[n-2]2^{n-2} + \ldots + d[1]2 + d[0]$, where $d[n-1] = 1$ and $d[i] = 0, 1$ $(i = 0, 1, ..., n-2)$. The binary method computes an ECDBL for every $d[i]$ and an ECADD if $d[i] \neq 0$. In average it requires $(n-1)$ ECDBLs and $(n-1)/2$ ECADDs. Because computing the inverse $-P$ of P is essentially free, we can relax the condition "binary" to "signed binary" $d = \sum_{i=0}^{n-1} d[i]2^i$, where $d[i] = -1, 0, 1$. It is called the signed binary method (or the addition-subtraction method). NAF offers a way to construct the addition-subtraction chain, which requires $(n-1)$ ECDBLs and $(n-1)/3$ ECADDs in average [IEEE] for an n-bit scalar multiplication. We denote the signed binary expression obtained by NAF as $d = \sum NAF(d)[i]2^i$. In the binary methods, points P and $-P$ are constant that we can set Z-coordinates of them to 1 for an efficiency reason.

Window Method: If we are allowed to use extra memory, the window methods enhance the efficiency of scalar multiplications by using a table of pre-computed points. In the following we assume the base point is not fixed. There are several types of window methods such as the 2^w-array window method, the sliding window method, the width-w NAF window method, and the combined fixed window methods [MvOV97,BHLM01]. As we will combine window methods and our proposed iterated ECDBL in section 3.2, we are particularly interested in the schemes that computes ECDBL successively.

We give a brief review of these methods. The 2^w-array window method with NAF is a direct extension of the signed binary method. It pre-computes $2P, \ldots, (2^w-1)P$ and in main loop it computes ECDBL w times and ECADD(Q, $\pm iP$) for $i \in \{0, \pm 1, \pm 2, \ldots, \pm(2^w-1)\}$ repeatedly.

The w-width sliding window method with NAF also use a pre-computed table (See Table 2). As the optimal window size of this method (in the sense of efficiency) for 160-bit scalar multiplications is $w = 4$ [dWMPW98], we assume $w = 4$. Then, the pre-computed windows are the following 10 forms ($\bar{1}$ denotes -1): $10\bar{1}0\,(6*P), 100\bar{1}\,(7*P), 1000\,(8*P), 1001\,(9*P), 1010\,(10*P)$, and their negatives which can be easily computed and are not stored in the table. Denote $e(xy) = 6, 7, 8, 9, 10$ for $xy = \bar{1}0, 0\bar{1}, 00, 01, 10$, respectively. We write 4 consecutive bits from the bit $NAF(d)[i]$ as 4-$NAF(d)[i]$. Denote ECDBL^k by the k-time iteration of ECDBL. In Table 2 we show the 4-width sliding window with NAF. In the pre-computation step, we compute 4 ECDBLs and 4 ECADDs. Then the five points $e(\bar{1}0)P, e(0\bar{1})P, e(00)P, e(01)P, e(10)P$ are converted to the affine coordinate and are stored in the table. In the evaluation step requires 160 ECDBLs and about 30 ECADDs on average [dWMPW98]. Therefore, the 4-width sliding window with NAF for 160 bits requires 164 ECDBLs and about 34 ECADDs on average, and 5 additional affine point storage.

Table 2. 4-width Sliding Window Method with NAF

```
Input: NAF(d), P, (NAF(d)[i], n)
Output: d*P
 Pre-computation
 1: 2P=ECDBL(P), 3P=ECADD(2P,P), 4P=ECDBL(2P), 6P=ECDBL(3P)
 2: 7P=ECADD(6P,P), 8P=ECDBL(4P), 9P=ECADD(8P,P), 10P=ECADD(9P,P)
 3: (e($\bar{1}$0)P,e(0$\bar{1}$)P,e(00)P,e(01)P,e(10)P) = Affine(6P,7P,8P,9P,10P)
 Evaluation
 1: Q = $\mathcal{O}$, i = n-1
 2: while i $\geq$ 3 do
 3:   if NAF(d)[i] = 0 then find the largest c
        with NAF(d)[i-j] = 0 for j $\leq$ c else c = 0
 4:   Q = ECDBL$^{c+4}$(Q), i = i-c
 5:     if 4-NAF(d)[i] = 10xy then Q = ECADD(Q,e(xy)P), i = i - 4
 6:     if 4-NAF(d)[i] = $\bar{1}0\bar{x}\bar{y}$ then Q = ECADD(Q,-e(xy)P), i = i - 4
 7: while i > 1 do
 8:   Q = ECDBL(Q)
 9:   if NAF(d)[i] = $\pm$1 then Q = ECADD(Q,$\pm$P)
10: return Affine(Q)
```

Next is the width-w NAF window method (See Table 3), which is an efficient window based scheme with a small table size. The width-w NAF represents the scalar d as $d = \sum NAF_w(d)[i]2^i$, where $|NAF_w(d)[i]| < 2^{w-1}$ and among any w consecutive coefficients, at most one is nonzero. In the pre-computation stage, we precompute the points $P, 3P, \ldots, (2^{w-1}-1)P$. In the evaluation stage we compute ECDBL and ECADD with the point from the table based on the $NAF_w(d)[i]$. It is known that the evaluation stage requires n ECDBLs and $n/(w+1)$ ECADDs on average [BHLM01]. For 160 bit cases, width 4 is optimal in the sense of efficiency and memory. The pre-computation of $3P, 5P$, and $7P$ requires 3 ECDBLs and 3 ECADDs and the conversions to the affine coordinate. In the evaluation stage we need 160 ECDBLs and 32 ECADDs. Therefore the width-4 NAF window method requires 163 ECDBLs and 35 ECADDs and 3 additional point storage.

3 Scalar Multiplication with SIMD Operations

This section provides faster scalar multiplication algorithms with SIMD operations. First, we establish the algorithms for computing addition formulas in the Jacobian coordinate with SIMD, and an algorithm for computing the k-iterated ECDBL with SIMD. Then scalar multiplication algorithms based on the signed binary method and the window-based methods are discussed.

3.1 Previous Algorithms

Recently Smart proposed a family of elliptic curves which is suitable for implementing with SIMD. However these curves are not fully-compatible with stan-

Table 3. width-4 NAF Window Method

```
Input: NAF_4(d), P, (NAF_4(d)[i], n)
Output: d*P
Pre-computation
1: 2P=ECDBL(P), 3P=ECADD(2P,P), 4P=ECDBL(2P)
2: 5P=ECADD(4P,P), 6P=ECDBL(3P), 7P=ECADD(6P,P)
3: (3P,5P,7P) = Affine(3P,5P,7P)
Evaluation
1: Q = $\mathcal{O}$
2: for i = n-1 to 0
3:   Q = ECDBL(Q)
4:     if NAF_4(d)[i] $\neq 0$ then Q = ECADD(Q,$\pm$NAF_4(d)[i]P)
5: return Affine(Q)
```

dardized elliptic curves (Weierstrass form). Aoki et al. proposed efficient algorithms for computing addition formulas on Weierstrass form curves with SIMD operations [AHKO01]. The computing times of algorithms of Aoki et al. with SIMD are $\texttt{ECADD}^{\mathrm{AHKO}}_{Z=1} = 5M + 1S$ and $\texttt{ECDBL}^{\mathrm{AHKO}} = 2M + 3S$.

3.2 New Algorithms

Actually, the algorithms by Aoki et al. compute the addition formulas efficiently, they implicitly assumed that the computing time for a multiplication by a constant is negligible. They did not discuss general case of ECADD (when $Z \neq 1$). Moreover, they did not include the cost of additions in algorithms. Generally computing additions are easier than multiplications and squaring. However, to make a minute comparison, the numbers of addition should be considered.

We show an improved version of parallelized algorithms without neglecting a multiplication by a constant. The concrete algorithms are summarized in Table 9 in appendix. The computing times for ECADD and ECDBL with SIMD are $\texttt{ECADD}^{\mathrm{new}} = 6M + 2S + 5A$ with 8 variables and $\texttt{ECDBL}^{\mathrm{new}} = 2M + 3S + 7A$ with 7 variables. If one of Z-coordinates of inputs is 1, $\texttt{ECADD}^{\mathrm{new}}$ is deduced to $\texttt{ECADD}^{\mathrm{new}}_{Z=1} = 4M + 2S + 6A$ with 8 variables. If $a = -3$, $\texttt{ECDBL}^{\mathrm{new}}$ is deduced to $\texttt{ECDBL}^{\mathrm{new}}_{a=-3} = 2M + 2S + 7A$ with 8 variables.

If we successively compute ECDBL k times, we call this operation as the k-iterated ECDBL. We can compute iterated ECDBL by computing ECDBL k times. However, the computing time may be enhanced if intermediate results are reused. Indeed, Itoh et al. proposed an efficient algorithm for the k-iterated ECDBL in the Jacobian coordinate [ITTTK99].

We parallelized their algorithm and optimized the computing time with SIMD operations. The proposed algorithm is in Table 10 in the appendix. The computing time for k-iterated ECDBL is $\texttt{kECDBL} = 2kM + (2k+1)S + 7kA$ with 8 variables. If we apply $\texttt{ECDBL}^{\mathrm{new}}$ k times, the computing time is $2kM + 3kS + 7kA$, which is slower than $\texttt{kECDBL}$ ($k \geq 2$). On the other hand, if $a = -3$, this

Table 4. Signed binary method with k-ECDBL

```
Input: NAF(d), P, (NAF(d)[i],n)
Output: d*P
1: Q = $\mathcal{O}$, i = n-1
2: while i $\geq$ 1 do
3:   if NAF(d)[i] = 0 then find the largest c
       with NAF(d)[i-j] = 0 for j $\leq$ c else c = 0
4:   Q = (c+1)-ECDBL(Q), i=i-c
5:     if NAF(d)[i] = $\pm$1 then Q=ECADD$^{new}_{Z=1}$(Q,$\pm$P), i = i - 1
6: return Affine(Q)
```

approach is not effective. That is, applying $\texttt{ECDBL}^{\mathrm{new}}_{a=-3}$ k times, which needs $2kM + 2kS + 7kA$, is slightly faster.

3.3 Signed Binary Methods

We are going to discuss the scalar multiplication algorithms in this and next sections. In this section, we propose non-window based algorithms, which is faster than the scheme using the formulas [AHKO01]. For a general purpose, we assume $a \neq -3$ in the following. But we assume the bit length of a scalar is 160. The comparison is measured by the time of multiplications, squarings, and inversions in the definition field K. We neglect computing times for additions (subtractions) in K.

First, we apply our proposed $\texttt{ECADD}^{\mathrm{new}}_{Z=1}$ and $\texttt{ECDBL}^{\mathrm{new}}$ to the signed binary method with NAF. The estimated computing times for both our method and the method in [AHKO01] are roughly equal to $(n-1)$ ECDBL + $n/3$ ECADD and more $3M + 1S + 1I$ for converting the coordinate from the Jacobian to the affine coordinate by computing $(X, Y, Z) \to (X/Z^2, Y/Z^3)$. The scheme of [AHKO01] requires $587.7M + 531.3S + 1I = 1042.7M$. Our proposed scheme requires $534.3M + 584.7S + 1I = 1032.1M$.

Next, we apply the k-iterated ECDBL to the signed binary chain. $\texttt{ECADD}^{\mathrm{new}}$ is computed if and only if the i-th bit of the NAF chain $NAF(d)[i]$ is not zero. Therefore, we can repeatedly apply k-iterated ECDBL for the consecutive zero sequences instead of k times ECDBLs. The algorithm is as follows:

Let us estimate the efficiency of the method. Koyama-Tsuruoka estimated the average length of c for NAF is 4/3 [KT92]. Therefore the averaged length k of k-iterated ECDBL is $7/3 = 2.33$ and the number of iterated ECDBL is 68.14 on average for 160-bit cases. The average number of ECADD is $159/3 = 53$. Thus the total efficiency is $532.5M + 492.7S + 1I = 956.7M$. Our proposed scheme using $\texttt{ECADD}^{new}$ and $\texttt{kECDBL}$ is about 10% faster than the previously fastest algorithm.

In order to confirm the appropriateness of our efficiency estimations, we made an experiment for 10,000 randomly selected 160-bit scalars. These results justify our estimations. We summarize the estimations and experimental results in Table 7, where we set $1S = 0.8M,\ 1I = 30M$ as in [OS00,IT02a].

Table 5. Comparison of the computing times of signed binary methods

	Estimation		Exp. Res.
NAF with $\texttt{ECADD}^{\mathrm{AHKO}}$, $\texttt{ECDBL}^{\mathrm{AHKO}}$	$587.7M + 531.3S + 1I$	$1042.7M$	$1044.4M$
NAF with $\texttt{ECADD}^{\mathrm{new}}_{Z=1}$, $\texttt{ECDBL}^{\mathrm{new}}$	$534.3M + 584.7S + 1I$	$1032.1M$	$1033.7M$
NAF with $\texttt{ECADD}^{\mathrm{new}}_{Z=1}$, $\texttt{kECDBL}$	$532.5M + 492.7S + 1I$	$\mathbf{956.7M}$	$\mathbf{947.8M}$

3.4 Window-Based Methods

In this section, we discuss the window-based methods, i.e. the sliding window method and the width-w NAF window method. We do not consider the window-based schemes that use the fixed base exponentiation, e.g. the combined fixed window method [MvOV97]. In the following, the formulas $\texttt{ECADD}^{\mathrm{new}}_{Z=1}$ and $\texttt{ECDBL}^{\mathrm{new}}$ will be abbreviated as $\texttt{ECADD}_{Z=1}$ and $\texttt{ECDBL}$, respectively.

The window-based methods consist of the pre-computation stage and the evaluation stage. In order to improve the efficiency of the evaluation stage, we employ $\texttt{ECADD}_{Z=1}$, namely Z-coordinate of points in the table are chosen 1. We only need one inversion to transform all points by Montgomery's trick, which converts k inversions into $3(k-1)$ multiplications and 1 inversion.

At first we estimate the efficiency of the 4-width sliding window method described in section 2. In the pre-computation stage, we need 4 $\texttt{ECDBL}$ and 4 $\texttt{ECADD}_{Z=1}$ for point computation and $27M + 5S + 1I$ for the transformation of the 5 points to the affine coordinate. The total efficiency of the pre-computation is $51M + 25S + 1I$. We need 160 $\texttt{ECDBL}$+ 30 $\texttt{ECADD}_{Z=1}$ for the evaluation stage and $3M + 1S + 1I$ for the conversion to the affine coordinate. Thus the 4-width sliding windows requires $494M + 566S + 2I = 1006.8$. Note that if we apply the $\texttt{ECADD}^{\mathrm{AHKO}}$ to the scheme it becomes $6M$ slower.

Here our proposed $\texttt{kECDBL}$ can reduce the computing time of step 3 in Table 3, if we apply it to the 4-width sliding window method with NAF, because the method computes ECDBL (at least) 4 times successively. If the largest integer c with $NAF(d)[i-l]$ for $j \le c$ is longer in step 3, the improvement becomes larger. Win et al. estimated $l = 4/3$ and the expected number of the ECADD is $\lfloor 160/(4+4/3) \rfloor = 30$ [dWMPW98]. In step 8 we compute 1 $\texttt{ECADD}_{Z=1}$ on average. Therefore in total we require 30 (5.33-ECDBL) + 31 $\texttt{ECADD}_{Z=1}$ in the evaluation stage. Thus the 4-width sliding windows using 4-iterated ECDBL requires $497.8M + 437.8S + 2I = 908.4M$ on average. Our proposed scheme is about 10% faster than the original 4-width sliding window method. We made an experiment for $10,000$ randomly chosen scalar d as in Section 3.4. Then we have the following results. The average length of l is 1.13 and the average call of $4+l$-$\texttt{ECDBL}$ is 30.28. The number of $\texttt{ECADD}$ that appeared in step 4 (or 5) is 29.45. The computation of $i < 3$ in step 8 and 9 requires 1.21$\texttt{ECDBL}$ + 0.79 $\texttt{ECADD}_{Z=1}$. Therefore in total we require 30.28 (5.13-$\texttt{ECDBL}$) + 29.45 $\texttt{ECADD}_{Z=1}$ + 1.21 $\texttt{ECDBL}$ + 0.79 $\texttt{ECADD}_{Z=1}$ $= 434M + 405S$ in the 1.22 evaluation stage. Thus the total efficiency of the 4-width sliding windows requires $488M + 432S + 2I = 893.6M$.

Next, let us consider the efficiency of the width-4 NAF window method in section 2. The pre-computed points are $3P, 5P$, and $7P$ as the affine coordi-

Table 6. width-4 NAF Window Method with k-ECDBL

Input: $\mathtt{NAF_4(d),\ P,\ (NAF_4(d)[i],n)}$
Output: `d*P`

```
Pre-computation
1: 2P=ECDBL(P), 3P=ECADD(2P,P), 4P=ECDBL(2P)
2: 5P=ECADD(4P,P), 6P=ECDBL(3P), 7P=ECADD(6P,P)
3: (3P,5P,7P) = Affine(3P,5P,7P)
Evaluation
1: Q = $\mathcal{O}$, i = n-1
2: while i $\geq$ 1 do
3:   if NAF$_4$(d)[i] = 0 then find the largest c
       with NAF$_4$(d)[i-j] = 0 for j $\leq$ c else c = 0
4:   Q = (c+1)-ECDBL(Q), i=i-c
5:     if NAF$_4$(d)[i] $\not=$ 0 then Q = ECADD$^{new}_{Z=1}$(Q,$\pm$NAF$_4$(d)[i]P), i = i-1
6: return Affine(Q)
```

nate. We compute them as $2P = \mathtt{ECDBL}(P), 3P = \mathtt{ECADD}_{Z=1}(2P, P)$, $4P = \mathtt{ECDBL}(2P), 5P = \mathtt{ECADD}_{Z=1}(4P, P)$, $6P = \mathtt{ECDBL}(3P)$, $7P = \mathtt{ECADD}_{Z=1}(6P, P)$, which require $18M + 15S$. We convert the points $3P, 5P, 7P$ to the affine coordinate, which requires $3(3M + 1S + I) = 15M + 3S + 1I$. The total efficiency of the pre-computation stage is $33M + 18S + 1I$. We need $159\mathtt{ECDBL} + 32\mathtt{ECADD}_{Z=1}$ for the evaluation stage and $3M + 1S + 1I$ for the conversion to the affine coordinate. Thus the width-4 NAF window method requires $482M + 560S + 2I = 990M$. Note that if we apply the $\mathtt{ECADD}^{\mathrm{AHKO}}$ to the scheme it becomes $6.4M$ slower.

We apply `kECDBL` to the width-4 NAF window method (See Table 6). In the evaluation stage, `ECADD` is computed if and only if the $NAF_4(d)[i] \not= 0$. Let c be the largest integer which satisfies $NAF_4(d)[i] = NAF_4(d)[i-1] = ... = NAF_4(d)[i-c] = 0$. The expected length of c is 2 [dWMPW98]. Thus we can improve its efficiency by repeatedly applying `kECDBL`. We estimate the efficiency of the proposed method. The pre-computation stage requires as large as the original scheme, namely $33M + 18S + 1I$. The expected length of c is 2, and thus we have to compute 53 times 3-`ECDBL` and 32 $\mathtt{ECADD}_{Z=1}$ on average, which is equivalent to $446M + 435S$. Therefore the total efficiency of the 4-width NAF window method is $482M + 454S + 2I = 905.2M$ and it requires additional 3 points storage. Our experiment result is $884.7M$ for $10,000$ randomly chosen 160-bit scalar d. Our proposed scheme is about 10% faster than the original width-4 NAF window method.

Consequently, we summarize the results that we have discussed above in the following table.

Table 7. Comparison of the computing times of window based schemes

	Estimation		Exp. Res.	Table Size
4-width SW with $\mathtt{ECADL}^{\mathrm{new}}$, $\mathtt{ECDBL}^{\mathrm{new}}$	$494M + 566S + 2I$	$1006.6M$	$993.6M$	5 points
4-width SW with $\mathtt{ECADL}^{\mathrm{new}}$, `kECDBL`	$497.8M + 437.8S + 2I$	**908.4M**	**893.6M**	5 points
width-4 NAF with $\mathtt{ECADL}^{\mathrm{new}}$, $\mathtt{ECDBL}^{\mathrm{new}}$	$482M + 560S + 2I$	$990.0M$	$986.0M$	3 points
width-4 NAF with $\mathtt{ECADL}^{\mathrm{new}}$, `kECDBL`	$482M + 454S + 2I$	**905.2M**	**884.7M**	3 points

4 Scalar Multiplication and Side Channel Attacks

The shorter key-length of ECC is suitable for implementing on low-power devices such as smart cards. The side channel attacks (SCA) [Koc96,KJJ99] are serious if there is a tight connection between secret information hidden in the device and the side channel information observed by an attacker. The simple power analysis (SPA) and the differential power analysis (DPA) are good examples of SCA. Implementers should take measures against each of them.

4.1 Countermeasures against SCA

There are three approaches to resist SPA. The first one uses the indistinguishable addition formula in the scalar multiplication [BJ02]. The second one uses the add-and-double-always method [Cor99] which masks the dependency between the scalar and the side channel information. In the algorithm, both an ECDBL and an ECADD are computed in every bit. Thus an adversary cannot guess the bit information of d. Möller's window-based method is in this category [Möl01]. The last countermeasure is the Montgomery's ladder [Mon87], which essentially resists SPA [OKS00,BJ02,FGKS02,IT02a,IT02b]. The x-coordinate-only addition formula, which is a modification of the standard addition formula, combined with the Montgomery's ladder offers a fast scalar multiplication. Previously fastest algorithm for computing a scalar multiplication is based on this method [IT02b].

Even if a scheme is SPA-resistant, it is not always DPA-resistant. Coron proposed a countermeasure by randomizing parameters of ECC to resist DPA [Cor99]. A key of Coron's idea for the Jacobian coordinate is as follows. Let $P = (X, Y, Z)$ be a base point in a projective coordinate. Then (X, Y, Z) equals to (r^2X, r^3Y, rZ) for all $r \in K$. By randomizing a base point with r before starting the scalar multiplication, the side information for the statistic analysis will be randomized. The other countermeasure against DPA was proposed by Joye-Tymen [JT01]. They use an isomorphism of an elliptic curve. The base point $P = (X, Y, Z)$ and the definition parameters a, b of an elliptic curve are randomized into its isomorphic classes like $(r^2X : r^3Y : Z)$ and r^4a, r^6b, respectively.

4.2 Previous Addition Formulas

Previously fastest algorithm for a scalar multiplication with a single processor is by Izu-Takagi [IT02b]. The computing time for each bit is $13M + 4S$ (effects of additions are not discussed).

In a parallelized computation, Izu-Takagi also proposed an algorithm which needs $T_M = 8M + 2S$ (addition are not discussed) for each bit [IT02a]. However, their parallelizaion architecture is based on MIMD (Multiple Instructions, Multiple Data) operations and it is hard to implement on smart cards. Recently, Fishcher et al. proposed a parallelized algorithm with SIMD for a scalar multiplication which needs $T_0 = 10M + 8A$ in each bit [FGKS02].

4.3 Proposed Algorithms

At present, parallel computation with SIMD operations on high-power CPU is easily realized as in the previous section. However SIMD operations are thought to be hard to realize on smart cards. In the case of ECC arithmetic, a co-processor CRYPTO2000 offers the SIMD operations [FGKS02]. The CRYPTO2000 has two modes; one is the 2K RSA mode and the other is the 1/2K ECC mode. In the ECC mode, the CRYPTO2000 is divided into two parallel calculation units. Our study is motivated by this architecture and this is why SCA-resistant scalar multiplication with SIMD operations is required on smart cards.

In this section, we propose two SCA-resistant algorithms for computing a scalar multiplication with SIMD operations. The first one is a direct application of $\texttt{ECADD}^{\text{new}}_{Z=1}$ and $\texttt{ECDBL}^{\text{new}}$ to the add-and-double-always method. The second one is based on a parallelization of Möller's SCA-resistant window method [Möl01], to which our `kECDBL` in Section 3.3 can be applied

Proposal 1: By applying $\texttt{ECDBL}^{\text{new}}_{Z=1}$ and $\texttt{ECADD}^{\text{new}}$ (see Section 3.2) to the add-and-double-always method, we can establish an SPA-resistant scalar multiplication and, in addition, establish a DPA-resistant algorithm with Joye-Tymen's countermeasure. The computing time for each bit is $T_1 = \texttt{ECADD}^{\text{new}}_{Z=1} + \texttt{ECDBL}^{\text{new}} = (4M + 2S + 5A) + (2M + 3S + 7A) = 6M + 5S + 12A$. No pre-computations are needed. With Coron's countermeasure, we cannot keep Z-coordinate of P to 1, it makes the scalar multiplication slower. Note we cannot apply the k-iterated ECDBL because ECADD and ECDBL are computed one after another.

Proposal 2: A key idea of Möller's algorithm [Möl01] is to establish an add-and-double-always computation in 2^w-array window method. Möller gave an excellent algorithm to compute coefficients of $d = \sum_{i=1}^{k} D[i]2^{wi}$ where $D[i] \in \{-2^w, \pm 1, \pm 2, \ldots, \pm(2^{w-1} - 1), 2^{w-1}\}$. Then in the 2^w-array method, ECADD and ECDBL are always computed because $D[i] \;/\!\!= 0$. In this scenario, ECDBL is computed w times successively and our w-iterated ECDBL can be applied. For n-bit scalar multiplications, we need $\lceil \frac{n}{w} \rceil - 1$ loops and in each loop we need one w-iterated ECDBL and one ECADD. The computing time for each loop is $T_2^{\text{C}}(w) = w\text{-}\texttt{ECDBL} + \texttt{ECADD}^{\text{new}} = (2w + 6)M + (2w + 3)S + (7w + 5)A$ with Coron's countermeasure, and $T_2^{\text{JT}}(w) = w\text{-}\texttt{ECDBL} + \texttt{ECADD}^{\text{new}}_{Z=1} = (2w + 4)M + (2w + 3)S + (7w + 6)A$ with Joye-Tymen's countermeasure.

In the pre-computation we have to compute $2P, \ldots, (2^{w-1}-1)P, 2^{w-1}P, 2^w P$. With Coron's countermeasure, we require

$$\begin{aligned} T^{\text{C}}_{2,\text{Pre}}(w) &= (2^{w-2} - 1)\texttt{ECADD}^{\text{new}} + 2^{w-2}\texttt{ECDBL}^{\text{new}} \\ &= (8 \cdot 2^{w-2} - 6)M + (5 \cdot 2^{w-2} - 2)S + (12 \cdot 2^{w-2} - 5)A. \end{aligned}$$

With Joye-Tymen's countermeasure, after computing 2^{w-1} points, we have to convert them to the affine coordinate. We require

$$\begin{aligned} T^{\text{JT}}_{2,\text{Pre}}(w) = {} & (2^{w-2} - 1)\texttt{ECADD}^{\text{new}}_{Z=1} + 2^{w-2}\texttt{ECDBL}^{\text{new}} \\ & + 3(2^{w-1} - 2)M + 1I + 3(2^{w-1} - 1)M + (2^{w-1} - 1)S \\ = {} & (18 \cdot 2^{w-2} - 13)M + (7 \cdot 2^{w-2} - 3)S + 1I + (13 \cdot 2^{w-2} - 6)A. \end{aligned}$$

Table 8. Comparison of the computing times of a scalar multiplication

	Countermeasure	Estimation ($n = 160$)		Table Size
[FGKS02]	JT	$1618M + 10S + 1I$	$1656.0M$	—
Proposal 1	JT	$961M + 798S + 1I$	$1629.4M$	—
Proposal 2	Coron ($w = 4$)	$578M + 449S + 1I$	$967.2M$	8 points
	Coron ($w = 5$)	$558M + 443S + 1I$	$\mathbf{942.4M}$	16 points
	JT ($w = 4$)	$559M + 467S + 2I$	$\mathbf{992.6M}$	8 points
	JT ($w = 5$)	$573M + 459S + 2I$	$1000.2M$	16 points

Comparison: We give a time comparison of 3 parallelized SCA-resistant algorithms, (1) non-window method based on [FGKS02], (2) non-window method (Proposal 1), and (3) window method (Proposal 2), in Table 8. The computing times for additions are neglected. The computing times of (1) and (2) are roughly same, while the times of (3) are much faster. For Coron's countermeasure $w = 5$ is optimal, while for Joye-Tymen's countermeasure $w = 4$ is optimal.

References

AHKO01. K.Aoki, F. Hoshino, T. Kobayashi and H.Oguro, "Elliptic Curve Arithmetic Using SIMD", *ISC2001*, LNCS 2200, pp.235-247, Springer-Verlag, 2001.

BJ02. E.Brier and M.Joye, "Weierstraß Elliptic Curves and Side-Channel Attacks", *PKC2002*, LNCS 2274, pp.335-345, Springer-Verlag, 2002.

BHLM01. M. Brown, D. Hankerson, J. Lopez, and A. Menezes, "Software Implementation of the NIST Elliptic Curves Over Prime Fields", *CT-RSA 2001*, LNCS 2020, pp.250-265, 2001.

Cor99. J.Coron, "Resistance against differential power analysis for elliptic curve cryptosystems", *CHES'99*, LNCS 1717, pp.292-302, Springer-Verlag, 1999.

CMO98. H.Cohen, A.Miyaji and T.Ono, "Efficient elliptic curve exponentiation using mixed coordinates", *Asiacrypt'98*, LNCS 1514, pp.51-65, Springer-Verlag, 1998.

dWMPW98. E. de Win, S. Mister, B. Preneel, and M. Wiener, "On the Performance of Signature Schemes Based on Elliptic Curves", *ANTS-III*, LNCS 1423, pp.252-266, 1998.

FGKS02. W. Fischer, C. Giraud, E. Knudsen, and J.-P. Seifert , "Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against Non-Differential Side-Channel Attacks", Cryptology ePrint Archive, 2002/007, IACR, 2002.

IEEE. IEEE P1363, Standard Specifications for Public-Key Cryptography, 2000.

ITTTK99. K.Itoh, M.Takenaka, N.Torii, S.Temma, and Y.Kurihara, "Fast Implementation of Public-Key Cryptography on DSP TMS320C6201", *CHES'99*, LNCS 1717, pp.61-72, 1999.

IT02a. T.Izu and T.Takagi, "A fast parallel elliptic curve multiplication resistant against side channel attacks", *PKC2002*, LNCS 2274, pp.280-296, 2002.

IT02b. T.Izu and T.Takagi, "A fast parallel elliptic curve multiplication resistant against side channel attacks", Technical Report of CACR, CORR 2002-03, University of Waterloo, 2002. Available from `http://www.cacr.math.uwaterloo.ca`

JT01. M.Joye and C.Tymen, "Protections against differential analysis for elliptic curve cryptography", *CHES2001*, LNCS 2162, pp.377-390, Springer-Verlag, 2001.

Koc96. C.Kocher, "Timing attacks on Implementations of Diffie-Hellman, RSA, DSS, and other systems", *Crypto'96*, LNCS 1109, pp.104-113, Springer-Verlag, 1996.

KJJ99. C.Kocher, J.Jaffe and B.Jun, "Differential power analysis", *Crypto'99*, LNCS 1666, pp.388-397, Springer-Verlag, 1999.

KT92. K.Koyama and Y.Tsuruoka, "Speeding up elliptic curve cryptosystems using a signed binary windows method", *Crypto'92*, LNCS 740, pp.345-357, Springer-Verlag, 1992.

MvOV97. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*, CRC Press, 1997.

Möl01. B.Möller, "Securing elliptic curve point multiplication against side-channel attacks", *ISC 2001*, LNCS 2200. p.324-334, Springer-Verlag, 2001.

Mon87. P.Montgomery, "Speeding the Pollard and elliptic curve methods for factorizations", *Math. of Comp.*, vol.48, pp.243-264, 1987.

NIST. National Institute of Standards and Technology, Recommended Elliptic Curves for Federal Government Use, in the appendix of FIPS 186-2,

OKS00. K.Okeya, H.Kurumatani and K.Sakurai, "Elliptic curves with the Montgomery form and their cryptographic applications", *PKC2000*, LNCS 1751, pp.446-465, Springer-Verlag, 2000.

OS00. K.Okeya and K.Sakurai, "Power analysis breaks elliptic curve cryptosystems even secure against the timing attack", *Indocrypt 2000*, LNCS 1977, pp.178-190, Springer-Verlag, 2000.

Sma01. N.Smart, "The Hessian form of an elliptic curve", *CHES2001*, LNCS 2162, pp.118-125, Springer-Verlag, 2001.

SEC. Standards for Efficient Cryptography Group (SECG), Specification of Standards for Efficient Cryptography. Available from `http://www.secg.org`

Appendix

The appendix describes the formulas discussed in this paper.

Table 9. Proposed algorithms for $\texttt{ECADD}^{\text{new}}$ and $\texttt{ECDBL}^{\text{new}}$

$\texttt{ECADD}^{\text{new}}$ (in 6M+2S+5A with 8 var.)

Input: $P_1 = (X_1, Y_1, Z_1),\ P_2 = (X_2, Y_2, Z_2)$
Output: $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$
$R_1 \leftarrow X_1,\ R_2 \leftarrow Y_1,\ R_3 \leftarrow Z_1$
$R_4 \leftarrow X_2,\ R_5 \leftarrow Y_2,\ R_6 \leftarrow Z_2$

1.	$R_7 \leftarrow R_3^2$	$R_8 \leftarrow R_6^2$
2.	$R_4 \leftarrow R_4 \times R_7$	$R_1 \leftarrow R_1 \times R_8$
3.	$R_2 \leftarrow R_2 \times R_6$	$R_5 \leftarrow R_3 \times R_5$
4.	$R_8 \leftarrow R_2 \times R_8$	$R_7 \leftarrow R_5 \times R_7$
5.	$R_2 \leftarrow R_4 - R_1$	$R_5 \leftarrow R_7 - R_8$
6.	$R_7 \leftarrow R_2^2$	$R_4 \leftarrow R_5^2$
7.	$R_1 \leftarrow R_1 \times R_7$	$R_7 \leftarrow R_2 \times R_7$
8.	$R_8 \leftarrow R_8 \times R_7$	$R_3 \leftarrow R_3 \times R_6$
9.	$R_4 \leftarrow R_4 - R_7$	$R_6 \leftarrow R_1 + R_1$
10.	$R_4 \leftarrow R_4 - R_6$	—
11.	$R_1 \leftarrow R_1 - R_4$	—
12.	$R_5 \leftarrow R_5 \times R_1$	$R_3 \leftarrow R_3 \times R_2$
13.	$R_7 \leftarrow R_5 - R_8$	—

$X_3 \leftarrow R_4,\ Y_3 \leftarrow R_7,\ Z_3 \leftarrow R_3$

$\texttt{ECDBL}^{\text{new}}$ (in 2M+3S+7A with 7 var.)

Input: $P_1 = (X_1, Y_1, Z_1), a$
Output: $P_4 = 2 * P_1 = (X_4, Y_4, Z_4)$
$R_1 \leftarrow X_1,\ R_2 \leftarrow Y_1,\ R_3 \leftarrow Z_1$

1.	$R_4 \leftarrow R_1^2$	$R_5 \leftarrow R_3^2$
2.	$R_6 \leftarrow R_2^2$	$R_5 \leftarrow R_5^2$
3.	$R_1 \leftarrow R_1 \times R_6$	$R_5 \leftarrow a \times R_5$
4.	$R_7 \leftarrow R_4 + R_4$	$R_5 \leftarrow R_4 + R_5$
5.	$R_5 \leftarrow R_5 + R_7$	$R_1 \leftarrow R_1 + R_1$
6.	$R_4 \leftarrow R_5^2$	$R_6 \leftarrow R_6^2$
7.	$R_1 \leftarrow R_1 + R_1$	$R_6 \leftarrow R_6 + R_6$
8.	$R_7 \leftarrow R_1 + R_1$	$R_6 \leftarrow R_6 + R_6$
9.	$R_4 \leftarrow R_4 - R_7$	$R_6 \leftarrow R_6 + R_6$
10.	$R_1 \leftarrow R_1 - R_4$	—
11.	$R_1 \leftarrow R_1 \times R_5$	$R_2 \leftarrow R_2 \times R_3$
12.	$R_1 \leftarrow R_1 - R_6$	$R_2 \leftarrow R_2 + R_2$

$X_4 \leftarrow R_4,\ Y_4 \leftarrow R_1,\ Z_4 \leftarrow R_2$

$\texttt{ECADD}^{\text{new}}_{Z=1}$ (in 4M+2S+6A with 8 var.)

Input: $P_1 = (X_1, Y_1, Z_1),\ P_2 = (X_2, Y_2, 1)$
Output: $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$
$R_1 \leftarrow X_1,\ R_2 \leftarrow Y_1,\ R_3 \leftarrow Z_1$
$R_4 \leftarrow X_2,\ R_5 \leftarrow Y_2$

1.	$R_6 \leftarrow R_3^2$	—
2.	$R_7 \leftarrow R_5 \times R_3$	$R_8 \leftarrow R_4 \times R_6$
3.	$R_4 \leftarrow R_8 - R_1$	—
4.	$R_8 \leftarrow R_6 \times R_7$	$R_3 \leftarrow R_3 \times R_4$
5.	$R_5 \leftarrow R_8 - R_2$	—
6.	$R_7 \leftarrow R_4^2$	$R_8 \leftarrow R_5^2$
7.	$R_4 \leftarrow R_4 \times R_7$	$R_1 \leftarrow R_1 \times R_7$
8.	$R_8 \leftarrow R_8 - R_4$	$R_6 \leftarrow R_1 + R_1$
9.	$R_8 \leftarrow R_8 - R_6$	—
10.	$R_1 \leftarrow R_1 - R_8$	—
11.	$R_1 \leftarrow R_5 \times R_1$	$R_2 \leftarrow R_2 \times R_4$
12.	$R_1 \leftarrow R_1 - R_2$	

$X_3 \leftarrow R_8,\ Y_3 \leftarrow R_1,\ Z_3 \leftarrow R_3$

$\texttt{ECDBL}^{\text{new}}_{a=-3}$ (in 2M+2S+7A with 8 var.)

Input: $P_1 = (X_1, Y_1, Z_1)$
Output: $P_4 = 2 * P_1 = (X_4, Y_4, Z_4)$
$R_1 \leftarrow X_1,\ R_2 \leftarrow Y_1,\ R_3 \leftarrow Z_1$

1.	$R_4 \leftarrow R_2^2$	$R_5 \leftarrow R_3^2$
2.	$R_4 \leftarrow R_4 + R_4$	$R_6 \leftarrow R_1 - R_5$
3.	$R_8 \leftarrow R_4 + R_4$	$R_7 \leftarrow R_1 + R_5$
4.	$R_1 \leftarrow R_1 \times R_8$	$R_6 \leftarrow R_6 \times R_7$
5.	$R_8 \leftarrow R_1 + R_1$	$R_7 \leftarrow R_6 + R_6$
6.	$R_6 \leftarrow R_6 + R_7$	—
7.	$R_4 \leftarrow R_4^2$	$R_7 \leftarrow R_6^2$
8.	$R_4 \leftarrow R_4 + R_4$	$R_7 \leftarrow R_7 + R_8$
9.	$R_1 \leftarrow R_1 - R_7$	—
10.	$R_1 \leftarrow R_1 \times R_6$	$R_2 \leftarrow R_2 \times R_3$
11.	$R_1 \leftarrow R_1 - R_4$	$R_2 \leftarrow R_2 + R_2$

$X_4 \leftarrow R_7,\ Y_4 \leftarrow R_1,\ Z_4 \leftarrow R_2$

Table 10. Proposed algorithm for k-iterated ECDBL ($\texttt{kECDBL}$)

k-$\texttt{ECDBL}$ (in $2kM + (2k+1)S + 7kA$ with 8 var.)

Input: $k, P_1 = (X_1, Y_1, Z_1), a$
Output: $P_4 = 2^k * P_1 = (X_4, Y_4, Z_4)$
$R_1 \leftarrow X_1,\ R_2 \leftarrow Y_1,\ R_3 \leftarrow Z_1$

1.	$R_4 \leftarrow R_1^2$	$R_5 \leftarrow R_3^2$	13. (repeat the following $k-1$ times)		
2.	$R_6 \leftarrow R_2^2$	$R_5 \leftarrow R_5^2$	13-1.	$R_3 \leftarrow R_4^2$	$R_5 \leftarrow R_1^2$
3.	$R_1 \leftarrow R_1 \times R_6$	$R_8 \leftarrow a \times R_5$	13-2.	$R_4 \leftarrow R_4 \times R_5$	$R_8 \leftarrow R_8 \times R_6$
4.	$R_7 \leftarrow R_4 + R_4$	$R_5 \leftarrow R_4 + R_8$	13-3.	$R_8 \leftarrow R_8 + R_8$	$R_7 \leftarrow R_3 + R_3$
5.	$R_5 \leftarrow R_7 + R_5$	$R_1 \leftarrow R_1 + R_1$	13-4.	$R_6 \leftarrow R_3 + R_7$	$R_4 \leftarrow R_4 + R_4$
6.	$R_4 \leftarrow R_5^2$	$R_6 \leftarrow R_6^2$	13-5.	$R_3 \leftarrow R_8 + R_6$	$R_7 \leftarrow R_4 + R_4$
7.	$R_1 \leftarrow R_1 + R_1$	$R_6 \leftarrow R_6 + R_6$	13-6.	$R_6 \leftarrow R_3^2$	$R_5 \leftarrow R_5^2$
8.	$R_7 \leftarrow R_1 + R_1$	$R_6 \leftarrow R_6 + R_6$	13-7.	$R_5 \leftarrow R_5 + R_5$	$R_4 \leftarrow R_7 + R_7$
9.	$R_4 \leftarrow R_4 - R_7$	$R_6 \leftarrow R_6 + R_6$	13-8.	$R_5 \leftarrow R_5 + R_5$	$R_4 \leftarrow R_6 - R_4$
10.	$R_1 \leftarrow R_1 - R_4$	—	13-9.	$R_6 \leftarrow R_5 + R_5$	$R_7 \leftarrow R_7 - R_4$
11.	$R_1 \leftarrow R_5 \times R_1$	$R_2 \leftarrow R_2 \times R_3$	13-10.	$R_7 \leftarrow R_3 \times R_7$	$R_2 \leftarrow R_1 \times R_2$
12.	$R_1 \leftarrow R_1 - R_6$	$R_2 \leftarrow R_2 + R_2$	13-11.	$R_1 \leftarrow R_7 - R_6$	$R_2 \leftarrow R_2 + R_2$

$X_4 \leftarrow R_4,\ Y_4 \leftarrow R_1,\ Z_4 \leftarrow R_2$

Further Results on Multiples of Primitive Polynomials and Their Products over GF(2)

Ayineedi Venkateswarlu[1,⋆] and Subhamoy Maitra[2]

[1] Dept. of Computer Sc. & Engineering, Indian Institute of Technology Madras
Chennai 600 036, India
ayineedi@yahoo.com

[2] Applied Statistics Unit, Indian Statistical Institute
203, B T Road, Calcutta 700 108, India
subho@isical.ac.in

Abstract. Recently the problem of analysing the multiples of primitive polynomials and their products has received a lot of attention. These primitive polynomials are basically the connection polynomials of the LFSRs (Linear Feedback Shift Registers) used in the stream cipher system. Analysis of sparse multiples of a primitive polynomial or product of primitive polynomials helps in identifying the robustness of the stream ciphers based on nonlinear combiner model. In this paper we first prove some important results related to the degree of the multiples. Earlier these results were only observed for small examples. Proving these results clearly identify the statistical behavior related to the degree of multiples of primitive polynomials or their products. Further we discuss a randomized algorithm for finding sparse multiples of primitive polynomials and their products. Our results clearly identify the time memory trade off for finding such multiples.

Keywords: Cryptanalysis, Galois Field, Nonlinear Combiner Model, Primitive Polynomials & Their Products, Polynomial Multiples, Stream Cipher.

1 Introduction

In the nonlinear combiner model of a stream cipher the outputs of several LFSRs are combined using a nonlinear Boolean function to produce cryptographically secure key stream (see [3,14,2] for more details). The connection polynomials of these LFSRs are polynomials over GF(2). For cryptographic robustness these polynomials are taken to be primitive [9] and it is expected that these primitive polynomials should be of high weight and also they should not have sparse multiples [13,1] (see also [8] and the references in this paper for current research on cryptanalysis on the nonlinear combiner model). The combining Boolean function should also have some properties to resist certain attack [14,15,1].

⋆ This work has been done as a part of M. Tech. (Computer Science) dissertation work at Indian Statistical Institute, Calcutta.

In this paper we concentrate on the sparse multiples of the connection polynomials corresponding to the LFSRs. Analysis of sparse multiples of primitive polynomials has received considerable attention recently [6,4,5,12]. In [5] it has been shown that given any primitive polynomial of degree d, it has exactly $N_{d,t} = (\binom{2^d-2}{t-2} - N_{d,t-1} - \frac{t-1}{t-2}(2^d - t + 1)N_{d,t-2})/(t-1)$ many t-nomial multiples (having constant term 1) with initial conditions $N_{d,2} = N_{d,1} = 0$. Moreover, it has been identified [5] that the *distribution of the degrees of t-nomial multiples* (having constant term 1) of a degree d primitive polynomial $f(x)$ is almost indistinguishable with the *distribution of maximum of the tuples having size* $(t-1)$ *in the range 1 to* $2^d - 2$. Further experiments in [12] have substantiated this claim and it has been observed that *average of square of the degrees of t-nomial multiples* and *average of square of the maximum of the tuples having size* $(t-1)$ are almost equal. A very interesting issue related to the trinomial multiples was observed in [12], which says that *average of square of the degrees of trinomial multiples* and *average of square of the maximum of the tuples having size* 2 are exactly equal. As a formal statement, *consider any primitive polynomial $f(x)$ of degree d. Consider that the degree of the trinomial multiples (having degree $\leq 2^d-2$) of $f(x)$ are $d_1, d_2, \ldots, d_{N_{d,3}}$. Then* $\sum_{i=1}^{N_{d,3}} d_i^2 = \frac{2}{3}(2^d-1)(3.2^{d-2}-1)N_{d,3}$. The observation has been made on experiments over primitive polynomials with small degrees 4 to 7 only. We here theoretically prove this result in Subsection 2.1.

Consider the nonlinear combiner model [14,15] which combines the outputs of several LFSRs using a nonlinear Boolean function. To study the robustness of such systems, it is important to analyse the sparse multiples of products of primitive polynomials (see [1,12] for details). Generally the degree of the primitive polynomials are taken to be coprime for generation of key stream having better cryptographic properties [9, Page 224]. Hence, if one can find sparse multiples of the product of primitive polynomials, then it is possible to launch cryptanalytic attacks on the nonlinear combiner model of the stream cipher (see [1] for a concrete description of such an attack).

In this direction the work in [12] concentrates on t-nomial multiples of products of primitive polynomials. Consider k different primitive polynomials $f_1(x)$, $f_2(x), \ldots, f_k(x)$ having degree $d_1, d_2, \ldots, d_k$ respectively, where $d_1, d_2, \ldots, d_k$ are pairwise coprime. It is observed [12] that the distribution of the degrees of t-nomial multiples (having constant term 1) of product of primitive polynomials is very close with the distribution of maximum of the tuples having size $(t-1)$. In [12] the following observations were made based on experiments. (1) The average of degree of the t-nomial multiples of $\prod_{r=1}^{k} f_r(x)$ is fixed and it is equal to $\frac{t-1}{t}\delta$, where δ is the exponent of $\prod_{r=1}^{k} f_r(x)$. (2) The average of the square of degree of the trinomial multiples of $\prod_{r=1}^{k} f_r(x)$ is fixed but not exactly equal to the estimated value $\frac{2}{3}\delta(\frac{3(\delta+1)}{4} - 1)$. We here settle both the problems with theoretical proofs in Subsection 2.2. With the work of [12] and the results proved in this paper, it is clear that the degree distribution of *multiples of primitive polynomials* and *multiples of products of primitive polynomials* are close. This assumption has also been considered earlier in [1, Page 581].

Though a lot of works are presented recently for analysing sparse multiples of primitive polynomials, algorithms to find them in practice has not been investigated properly. This we discuss in Section 3. Recent studies [5] identify that given a primitive polynomial $f(x)$ of degree d, the expected number of t-nomial multiples less than or equal to degree c is $\binom{c}{t-1} N_{d,t} / \binom{2^d-2}{t-1}$. Given such a c, we here analyse a randomized algorithm and show that the algorithm needs s iteration to produce a t-nomial multiple having degree $\leq c$ of $f(x)$ with probability as high as $1 - e^{-u}$, where $s\binom{c}{t-1} N_{d,t} / (\binom{c}{t-2}\binom{2^d-2}{t-1}) = u$. Moreover, if we consider sparse multiples (i.e., $t \leq 10$), then given such a c, the algorithm takes s steps ($cs \approx 2^{d+2}$) and provides a solution with probability 0.95. Further we analyse the time memory trade off for this algorithm. In this direction we also point out how the algorithms for Discrete Log Problem can be suitably used here. We also mention how one can use similar strategy to get sparse multiples of products of primitive polynomials.

Let us now briefly discuss a few basic concepts. The field of 2 elements is denoted by $GF(2)$. $GF(2^d)$ denotes the extension field of dimension d over $GF(2)$. A polynomial is irreducible over a field if it is not the product of two polynomials of lower degree in the field. An irreducible polynomial of degree d is called primitive if its roots are the generators of the field $GF(2^d)$. The exponent of the polynomial $f(x)$ (having degree $d \geq 1$, with $f(0) = 1$) is $e \leq 2^d - 1$, which is the least positive integer such that $f(x)$ divides $x^e - 1$. For primitive polynomials $e = 2^d - 1$. By a t-nomial we refer to a polynomial with t distinct non zero terms. For more details on finite fields, the reader is referred to [10,9].

2 On Degrees and Their Squares of t-Nomial Multiples

2.1 Trinomial Multiples of Primitive Polynomials: Square of Degrees

In [5], the distribution of the degrees for the t-nomial multiples (having constant term 1) of primitive polynomials has been discussed. Given any primitive polynomial $f(x)$ of degree d, it is clear that $f(x)$ has $N_{d,t}$ number of t-nomial multiples having degree $\leq 2^d - 2$. From cryptanalytic point of view, it is an important question that how many t-nomial multiples are there having degree less than or equal to some c. Since, this result is not settled, in [5], an estimation has been used. In [5], any t-nomial multiple $1 + x^{i_1} + x^{i_2} + \ldots + x^{i_{t-2}} + x^{i_{t-1}}$ has been interpreted as the $(t-1)$-tuple $< i_1, i_2, \ldots, i_{t-2}, i_{t-1} >$. It was also empirically justified using experimental results [5] that by fixing $f(x)$, if one enumerates all the $N_{d,t}$ different $(t-1)$ tuples, then the distribution of the tuples seems random. Moreover, the distribution of the degrees of the t-nomial multiples seems very close with the distribution of maximum value of each of the ordered tuples $< i_1, i_2, \ldots, i_{t-2}, i_{t-1} >$ with $1 \leq i_1 < i_2 < \ldots < i_{t-2} < i_{t-1} \leq 2^d - 2$.

To analyse the degree distribution of these t-nomial multiples, the random variate X is considered in [5], which is $\max(i_1, i_2, \ldots, i_{t-2}, i_{t-1})$, where $1 + x^{i_1} + x^{i_2} + \ldots + x^{i_{t-2}} + x^{i_{t-1}}$ is a t-nomial multiple of $f(x)$. There are $N_{d,t}$ such multiples. The mean value [5] of the distribution of X is $\frac{t-1}{t}(2^d - 1)N_{d,t}$ divided by $N_{d,t}$, i.e., $\overline{X} = \frac{t-1}{t}(2^d - 1)$. On the other hand, consider all the $(t-1)$-tuples $< i_1, i_2, \ldots, i_{t-2}, i_{t-1} >$ in the range 1 to $2^d - 2$. There are $\binom{2^d-2}{t-1}$ such tuples. Each tuple is in ordered form such that $1 \leq i_1 < i_2 < \ldots < i_{t-2} < i_{t-1} \leq 2^d - 2$. Consider the random variate Y which is $\max(i_1, i_2, \ldots, i_{t-2}, i_{t-1})$. It has been shown in [5] that the mean of this distribution is $\overline{Y} = \frac{t-1}{t}(2^d - 1)$.

Thus, given any primitive polynomial $f(x)$ of degree d, the average degree of its t-nomial multiples with degree $\leq 2^d - 2$ is equal to the average of maximum of all the distinct $(t-1)$ tuples form 1 to $2^d - 2$. With this result and experimental observations, the work of [5] assumes that the distributions X, Y are very close. Further experimental results have been presented in [12] to strengthen the claim of [5] that the distributions X, Y are very close. In this direction, it has been shown in [12] that in terms of average of squares, the distributions X, Y are very close. The average of squares of the values in Y have been calculated in [12] as $\frac{t-1}{t}(2^d - 1)(\frac{t2^d}{t+1} - 1)$ and it has been shown experimentally that the average of values in X are very close to that of Y. In [12], it has been observed that for $t = 3$, the average of the squares of the elements of distribution Y and the average of the squares of the degrees of trinomial multiples are same for all the experiments, which is $\frac{2}{3}(2^d - 1)(3.2^{d-2} - 1)$. We theoretically prove it here.

Theorem 1. *Consider any primitive polynomial $f(x)$ of degree d. Consider that the degree of the trinomial multiples (having degree $\leq 2^d - 2$) of $f(x)$ are $d_1, d_2, \ldots, d_{N_{d,3}}$. Then $\sum_{s=1}^{N_{d,3}} d_s^2 = (2/3)(2^d - 1)(3.2^{d-2} - 1)N_{d,3}$.*

Proof. Consider a trinomial multiple of $f(x)$ of the form $x^i + x^j + 1$, where $i > j$. Let $e = 2^d - 1$. Let $i \neq 2(2^d - 1)/3, j \neq (2^d - 1)/3$. Then $x^{(e-i)+j} + x^{e-i} + 1$ and $x^{e-j} + x^{i-j} + 1$ are two more distinct trinomial multiples of $f(x)$ (multiplying $x^i + x^j + 1$ by x^{e-i} and x^{e-j} respectively). Now, consider the difference $(i^2 - j^2) + ((e - i + j)^2 - (e - i)^2)) + ((e - j)^2 - (i - j)^2)$, which is equal to e^2. Further take the case $i = 2(2^d - 1)/3, j = (2^d - 1)/3$, when d is even. In that case all the three trinomials generated in the above manner are same. Thus we will only consider one difference, $(2(2^d - 1)/3)^2 - ((2^d - 1)/3)^2 = e^2/3$.

Let the trinomial multiples (having degree $< e$) of $f(x)$ be $x^{i_s} + x^{j_s} + 1$, for $s = 1, \ldots, N_{d,3}$. We will consider $\sum_{s=1}^{N_{d,3}}(i_s^2 - j_s^2)$. If d is odd we will get $N_{d,3}/3$ different groups each contributing e^2 in this sum. If d is even, we will get $(N_{d,3} - 1)/3$ different groups each contributing e^2 in this sum except one term which contributes $e^2/3$ when $i_s = 2(2^d - 1)/3, j_s = (2^d - 1)/(3)$.

Thus, $\sum_{s=1}^{N_{d,3}}(i_s^2 - j_s^2) = N_{d,3}e^2/3$. Now add $\sum_{s=1}^{N_{d,3}}(i_s^2 + j_s^2)$ in both sides. Then $2\sum_{s=1}^{N_{d,3}} i_s^2 = N_{d,3}e^2/3 + \sum_{s=1}^{N_{d,3}}(i_s^2 + j_s^2)$.

Note that, considering the values of i_s, j_s for all s we basically get all the integers in the range 1 to $e - 1$. Thus, $\sum_{s=1}^{N_{d,3}}(i_s^2 + j_s^2) = 1^2 + 2^2 + \ldots + (e-1)^2$. We already know [4] that $N_{d,3} = 2^{d-1} - 1$. Simplifying, we get $\sum_{s=1}^{N_{d,3}} i_s^2 = (2/3)(2^d - 1)(3.2^{d-2} - 1)N_{d,3}$. □

Theorem 1 proves the observation of [12]. This is now theoretically proved that for $t = 3$, the average of squares of the values in Y, i.e., $\frac{2}{3}(2^d - 1)(\frac{3\cdot 2^d}{4} - 1)$ is exactly equal to the average of square of the values in X.

2.2 Degrees and Square of Degrees for t-Nomial Multiples of Products of Primitive Polynomials

Consider k many primitive polynomials $f_1(x), f_2(x), \ldots, f_k(x)$ having degrees $d_1, d_2, \ldots, d_k$ respectively. Further, the degrees are pairwise coprime. We here follow the notations of [12]. To analyse the degree distribution of these t-nomial multiples of the products of primitive polynomials, let us consider the random variate $X^{(d_1,\ldots,d_k),t}$, which is $\max(I_1, \ldots, I_{t-1})$, where $x^{I_1} + x^{I_2} + \ldots + x^{I_{t-1}} + 1$ is a t-nomial multiple of $f_1(x)f_2(x)\ldots f_k(x)$. Let $\delta = (2^{d_1}-1)(2^{d_2}-1)\ldots(2^{d_k}-1)$. On the other hand, consider all the $(t-1)$-tuples $< I_1, \ldots, I_{t-1} >$, in the range 1 to $\delta - 1$. There are $\binom{\delta-1}{t-1}$ such tuples. Consider the random variate $Y^{(d_1,\ldots,d_k),t}$, which is $\max(I_1, \ldots, I_{t-1})$, where $< I_1, \ldots, I_{t-1} >$ is any ordered t-tuple from the values 1 to $\delta - 1$. With some experimental results, in [12], it was mentioned that the distributions $X^{(d_1,\ldots,d_k),t}$, $Y^{(d_1,\ldots,d_k),t}$ are very close. Based on experimental results, the following two observations were made in [12]. (1) The average of degree of the t-nomial multiples of $\prod_{r=1}^{k} f_r(x)$ is fixed and it is equal to $\frac{t-1}{t}\delta$, where δ is the exponent of $\prod_{r=1}^{k} f_r(x)$. (2) The average of the square of degree of the trinomial multiples of $\prod_{r=1}^{k} f_r(x)$ is fixed but not exactly equal to the estimated value $\frac{2}{3}\delta(\frac{3(\delta+1)}{4} - 1)$. We here prove these theoretically. First we present a technical result.

Lemma 1. *Let $f(x)$ be a polynomial over GF(2) having degree d and exponent e and $1+x$ does not divide $f(x)$. Let the number of t-nomial multiples (with degree $< e$ and constant term 1) of $f(x)$ be N_t^f. Then $\frac{N_t^f}{t} = \frac{N_{e-t}^f}{e-t}$, where $2 < t < e-2$.*

Proof. Note that $f(x)$ divides $1 + x^e$. Since $1 + x$ does not divide $f(x)$, $f(x)$ divides $\frac{1+x^e}{1+x}$, i.e., $f(x)$ divides $1 + x + x^2 + \ldots + x^{e-1}$. This is the e-nomial multiple of $f(x)$. Whenever $x^{i_1} + x^{i_2} + \ldots + x^{i_t}$ (constant term 0) is a multiple of $f(x)$ (here $1 \le i_1 < i_2 < \ldots < i_t < e$), adding with $1 + x + x^2 + \ldots + x^{e-1}$, we will get an $(e-t)$-nomial multiple $1 + \sum_{i=1, i \neq i_1, i_2, \ldots, i_t}^{e-1} x^i$ (having constant term 1) of $f(x)$.

We will count the number of such multiples of $f(x)$, which is equal to the number of $(e-t)$-nomials. Consider a t-nomial multiple $x^{j_1} + x^{j_2} + \ldots + x^{j_{t-1}} + 1$ of $f(x)$. Multiplying it by x^j for $0 \le j < e$, we will get t many t-nomial multiples having constant term 1 and $(e-t)$ many multiples of the form $x^{i_1} + x^{i_2} + \ldots + x^{i_t}$, (having constant term 0) where $1 \le i_1 < i_2 \ldots i_t < e$. Considering any one of these t many t-nomials (having constant term 1) will produce the same set of $(e-t)$ many $(e-t)$-nomial multiples. So, t many t-nomials giving $(e-t)$ many $(e-t)$-nomials and vice versa. Hence, we get $\frac{N_t^f}{t} = \frac{N_{e-t}^f}{e-t}$. □

Let us now present the following theorem.

Theorem 2. *Consider a polynomial $f(x)$ over GF(2) with exponent e such that $1 + x$ does not divide $f(x)$. Let the number of t-nomial multiples (with degree $< e$ and constant term 1) of f be N_t^f. Then the sum of the degrees of all it's t-nomial multiples with degree $< e$ is $\frac{t-1}{t} e N_t^f$.*

Proof. We have $1+x$ does not divide $f(x)$. Consider each t-nomial multiple of degree $\hat{d}_s$, where $1 \leq s \leq N_t^f$. Now multiply each t-nomial by x^i, for $1 \leq i \leq (e-\hat{d}_s-1)$, we will get multiples of the form $x^{i_1}+x^{i_2}+\ldots+x^{i_t}$, where $1 \leq i_1 < i_2 < \ldots < i_t < e$. Thus each t-nomial will provide $(e-\hat{d}_s-1)$ many multiples of the above form and observe that these are distinct. Similar to proof of Lemma 1, $\sum_{s=1}^{N_t^f}(e-\hat{d}_s-1)$ gives the count of $(e-t)$-nomial multiples. Moreover, from the proof of Lemma 1, we will get $N_{e-t}^f = \frac{e-t}{t}N_t^f$, i.e., $\sum_{s=1}^{N_t^f}(e-\hat{d}_s-1) = \frac{e-t}{t}N_t^f$. Hence $\sum_{s=1}^{N_t^f}\hat{d}_s = \left(e-1-\frac{e-t}{t}\right)N_t^f = \frac{t-1}{t}eN_t^f$. □

Corollary 1. *Consider k many primitive polynomials $f_1(x), f_2(x), \ldots, f_k(x)$ having degrees $d_1, d_2, \ldots, d_k$ respectively (the degrees are pairwise coprime). The average of degree of the t-nomial multiples (having degree $< \delta$) of $\prod_{r=1}^k f_r(x)$ is fixed and it is equal to $\frac{t-1}{t}\delta$, where δ is the exponent of $\prod_{r=1}^k f_r(x)$.*

Proof. Let $f(x) = \prod_{r=1}^k f_r(x)$. Since each $f_r(x)$ is a primitive polynomial of degree d_r, all the conditions of Theorem 2 are satisfied. So, $\frac{\sum_{s=1}^{N_t^f}\hat{d}_s}{N_t^f} = \frac{t-1}{t}\delta$. □

Hence, we prove that the average of the values in distributions $X^{(d_1,\ldots,d_k),t}$, and $Y^{(d_1,\ldots,d_k),t}$ are same which was presented as an observation in [12]. Next we consider the square of the degrees of trinomial multiples of $\prod_{r=1}^k f_r(x)$.

Theorem 3. *Take k many primitive polynomials $f_1(x), f_2(x), \ldots, f_k(x)$ over GF(2) having degrees $d_1, d_2, \ldots, d_k$ (pairwise coprime) and exponents $e_r = 2^{d_r}-1$, for $1 \leq r \leq k$. Then sum of squares of degrees of trinomial multiples of $f(x) = f_1(x)f_2(x)\ldots f_k(x)$ with degree $< e = e_1e_2\ldots e_k$ is $\frac{e^2}{6}2^{k-1}\prod_{r=1}^k(2^{d_r-1}-1) + \frac{(e-1)e(2e-1)}{12} + \frac{1}{2}\sum_{r=1}^{k-1}\sum_{A_r \subset \{e_1,e_2,\ldots,e_k\}}[(-1)^r(\prod_{e_j \in A_r} e_j{}^2)(\sum_{l=1}^{e/\prod_{e_j \in A_r} e_j - 1} l^2)]$ where $|A_r| = r$.*

Proof. Similar to proof of Theorem 1, considering all the trinomials $x^{i_s}+x^{j_s}+1$ of $f(x)$ with $1 \leq j < i < e$ for $1 \leq s \leq N_3^f$, we have $2\sum_{s=1}^{N_3^f} i_s{}^2 = \frac{N_3^f}{3}e^2 + \sum_{s=1}^{N_3^f}(i_s{}^2 + j_s{}^2)$.

Now we will see the possible values for i_s, j_s in the range $[1, e-1]$. Consider a trinomial multiple $x^i + x^j + 1$, where $1 \leq j < i < e$, of $f(x)$. It is important to see that this is not exactly similar to that of the proof of Theorem 1.

Note that, $x^{i \bmod e_r} + x^{j \bmod e_r} + 1$ is a trinomial multiple of $f_r(x)$, for $1 \leq r \leq k$ except the following case. If $i \bmod e_r = 0$ or $j \bmod e_r = 0$, then $x^{i \bmod e_r} + x^{j \bmod e_r} + 1$ is not a trinomial multiple of $f_r(x)$.

On the other hand, consider $x^i + 1$, where $1 \leq i < e$ and $i \not\equiv 0 \bmod e_r$, for all $r = 1, 2, \ldots, k$. Since $f_r(x)$ is primitive polynomial, for each $x^{i \bmod e_r} + 1$, where $1 \leq r \leq k$, we will get $x^{i \bmod e_r} + 1 \equiv x^{l_r} (\bmod f_r(x))$, where $1 \leq l_r < e_r$, i.e., $x^{i \bmod e_r} + x^{l_r} + 1$ is a trinomial multiple of $f_r(x)$. By using Chinese remainder theorem [7, Page 53], we get a unique integer $l \bmod e$, where $l \equiv l_r \bmod e_r$, for $1 \le r \le k$, as e_r's are pairwise coprime.

Hence, we have to discard the cases where, $1 \le l < e$ and $l \equiv 0 \bmod e_r$, for any r, $1 \le r \le k$. Then $\sum_{s=1}^{N_3^f}({i_s}^2 + {j_s}^2) = \sum_{i=1}^{e-1} i^2 - \sum_{x \in S} x$, where $S = \{l^2 : 1 \le l < e \text{ and } l \equiv 0 \bmod e_r, \text{ for any } r, 1 \le r \le k\}$.

Consider the sets $S_r = \{e_r^2, (2 \cdot e_r)^2, \ldots, ((\frac{e}{e_r} - 1) \cdot e_r)^2\}$, for $1 \le r \le k$. Observe that $\cup_{r=1}^{k} S_r = S$. We now calculate $\sum_{x \in S} x$ using inclusion and exclusion principle.

Take $n_1, n_2, \ldots, n_r$ as distinct integers in the range $[1, k]$. Now we consider $\cap_{q=1}^{r} S_{n_q}$, which contains $\prod_{q=1}^{r} {e_{n_q}}^2, 2^2 \cdot \prod_{q=1}^{r} {e_{n_q}}^2, \ldots, (e/\prod_{q=1}^{r} e_{n_q} - 1)^2 \cdot \prod_{q=1}^{r} {e_{n_q}}^2$. Hence, $\sum_{x \in \cap_{q=1}^{r} S_{n_q}} x = (\prod_{q=1}^{r} {e_{n_q}}^2)(\sum_{l=1}^{(e/\prod_{q=1}^{r} e_{n_q} - 1)} l^2)$. Finally, for $|A_r| = r$. $\sum_{x \in S} x = \sum_{x \in \cup_{r=1}^{k} S_r} x =$

$\sum_{r=1}^{k-1} \sum_{A_r \subset \{e_1, e_2, \ldots, e_k\}} [(-1)^{r+1} (\prod_{e_j \in A_r} {e_j}^2)(\sum_{l=1}^{(e/\prod_{e_j \in A_r} e_j) - 1} l^2)]$. So,

$2\sum_{s=1}^{N_3^f} {i_s}^2 = \frac{N_3^f}{3} e^2 + \sum_{s=1}^{N_3^f}({i_s}^2 + {j_s}^2) = \frac{N_3^f}{3} e^2 + \sum_{i=1}^{e-1} i^2 - \sum_{x \in S} x$. Hence, for $|A_r| = r$. $\sum_{s=1}^{N_3^f} {i_s}^2 = \frac{N_3^f}{6} e^2 + \frac{(e-1)e(2e-1)}{12} +$

$\frac{1}{2} \sum_{r=1}^{k-1} \sum_{A_r \subset \{e_1, e_2, \ldots, e_k\}} [(-1)^{r} (\prod_{e_j \in A_r} {e_j}^2)(\sum_{l=1}^{(e/\prod_{e_j \in A_r} e_j) - 1} l^2)]$.

From [12], we have the exact formula for the number of trinomial multiples (having degree $< e$) of $f(x)$, which is $2^{k-1} \prod_{r=1}^{k} (2^{d_r - 1} - 1)$ and this is the value of N_3^f. Hence the proof. □

3 Algorithm to Find Sparse Multiples

We start with the following simple algorithm which provides the least degree t-nomial multiple of a degree d primitive polynomial $f(x)$.

Algorithm 1 *Inputs : (1) a primitive polynomial $f(x)$ of degree d and its root α, (2) the value t, for the t-nomial multiple, (3) an integer $c \le 2^d - 2$. We need a t-nomial multiple having degree $\le c$.*

```
1. for $i = d + 1$ to $c$
   (a) Choose all possible sets of $t - 2$ distinct integers $i_1, \ldots, i_{t-2}$ in the range
       $[1, \ldots, i - 1]$.
   (b) If $f(x)$ divides $x^i + x^{i_1} + \ldots + x^{i_{t-2}} + 1$, then report it and terminate.
```

This will provide the lowest degree t-nomial multiple of $f(x)$. The complexity of this algorithm is $\sum_{j=d+1}^{c} \binom{j}{t-2}$. It is expected that [5] there is at least one t-nomial multiple below the degree $2^{\frac{d}{t-1} + \log_2(t-1)}$. Thus to analyse the complexity we can take $c = 2^{\frac{d}{t-1} + \log_2(t-1)}$. A rough approximation shows that the algorithm needs $O(2^d)$ steps. For large d, it is almost impossible to run this algorithm.

That is why we analyse a randomized algorithm which provides t-nomial multiples at much lower complexity. Consider a primitive polynomial $f(x)$ of degree d. Let α be a root of $f(x)$. Consider that we choose $(t-2)$ distinct integers $i_1, \ldots, i_{t-2}$ in the range 1 to c uniformly at random where $c < 2^d$. It is clear, that $1 + \alpha^{i_1} + \ldots + \alpha^{i_{t-2}}$ must be equal to some $\alpha^{i_{t-1}}$ for $0 \leq i_{t-1} \leq 2^d - 2$. Thus, $1 + x^{i_1} + \ldots + x^{i_{t-2}} + x^{i_{t-1}}$ will be a multiple of $f(x)$. Note that, if $i_{t-1} \notin \{0, i_1, \ldots, i_{t-2}\}$, then $1 + x^{i_1} + \ldots + x^{i_{t-2}} + x^{i_{t-1}}$ will be a t-nomial multiple of $f(x)$. Moreover, if $i_{t-1} \leq c$, we get a t-nomial multiple (of $f(x)$) with degree $\leq c$. We first present the randomized algorithm and analyse it next.

Algorithm 2 *Inputs as in Algorithm 1.*

```
1. Take $(t-2)$ distinct integers $i_1, \ldots, i_{t-2}$ form $[1, \ldots, c]$ uniformly at random.
2. Find out $i_{t-1}$, where, $1 + \alpha^{i_1} + \ldots + \alpha^{i_{t-2}} = \alpha^{i_{t-1}}$.
3. If $i_{t-1} \notin \{0, i_1, \ldots, i_{t-2}\}$ and $i_{t-1} \leq c$, then report $1+x^{i_1}+\ldots+x^{i_{t-2}}+x^{i_{t-1}}$
   and terminate[1]. Else go to step 1.
```

In [1, Page 580], similar algorithm has been discussed. In the actual implementation of the algorithm [1, Page 580], an array of length 2^d is required. That is the reason the algoithm takes just a single step to check whether $i_{t-1} \leq c$. However, an array of length 2^d is not possible to manage in practical computer systems if $d \geq 40$. If c is as large as 2^d, then the multiple will be of very high degree and the cryptanalytic attack will not succeed as the degree of the multiple should be of the order of (approximately half) the length of available cipher text. Thus, we need to consider c much lower than 2^d. We present Algorithm 3 (see later), which gives t-nomial multiples, even if $d \geq 40$, for c much lower than 2^d.

Now the question is what is the expected number of iterations in Algorithm 2 before it terminates. For this we present a series of technical results. Note that here $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots +$ upto ∞ (unlike the previous sections where e was used as the exponent of a polynomial).

Proposition 1. *As described in Algorithm 2 , the probability of getting a t-nomial multiple in a single step is* $\pi = 1 - e^{-\binom{c}{t-1} N_{d,t} / (\binom{c}{t-2}\binom{2^d-2}{t-1})}$.

Proof. Note that the expected number [5] of t-nomial multiples having degree $\leq c$ is $k = \binom{c}{t-1} N_{d,t} / \binom{2^d-2}{t-1}$. There are $\binom{c}{t-2}$ distinct $(t-2)$-tuples upto c. Thus $\pi = 1 - (1 - 1/\binom{c}{t-2})^k$. Now we are interested about the term $(1 - 1/\binom{c}{t-2})^k$. Note that, for large c, and $t \geq 3$, $1/\binom{c}{t-2}$ is very small, and it tends to 0. So we can apply the result $\lim_{x \to 0}(1+x)^{\frac{1}{x}} = e$. Note that, $k = \binom{c}{t-1} N_{d,t} / \binom{2^d-2}{t-1} = \binom{c}{t-2} r$, where $r = \binom{c}{t-1} N_{d,t} / (\binom{c}{t-2}\binom{2^d-2}{t-1})$. Thus, $(1 - 1/\binom{c}{t-2})^k$ tends to e^{-r}, i.e., $e^{-\binom{c}{t-1} N_{d,t} / (\binom{c}{t-2}\binom{2^d-2}{t-1})}$. Hence, $\pi = 1 - e^{-\binom{c}{t-1} N_{d,t} / (\binom{c}{t-2}\binom{2^d-2}{t-1})}$. □

[1] Note that, if the step 3 in Algorithm 2 produces $i_{t-1} \in \{0, i_1, \ldots, i_{t-2}\}$, then we get a $(t-2)$-nomial multiple (having degree $\leq c$) of $f(x)$.

Corollary 2. *As described in Algorithm 2, the probability of getting a t-nomial multiple in s steps is* $p = 1 - e^{-s\binom{c}{t-1}N_{d,t}/(\binom{c}{t-2}\binom{2^d-2}{t-1})}$.

Proof. We have, $p = 1-(1-\pi)^s$. Using Proposition 1, we get the result. □

Theorem 4. *Consider a primitive polynomial $f(x)$ of degree d. Algorithm 2 needs s iteration to produce a t-nomial multiple having degree $\leq c$ of $f(x)$ with probability as high as $1-e^{-u}$, where $s\binom{c}{t-1}N_{d,t}/(\binom{c}{t-2}\binom{2^d-2}{t-1}) = u$.*

For practical purposes we need sparse multiples of primitive polynomials. In this direction, using some specific range of values, we try to approximate the complicated expression presented in Theorem 4. Our motivation is to get p very close to 1 for successful outcome of the algorithm. Choosing $p = 0.95$, we get $e^{-s\binom{c}{t-1}N_{d,t}/(\binom{c}{t-2}\binom{2^d-2}{t-1})} = 0.05$. Thus, $s\binom{c}{t-1}N_{d,t}/(\binom{c}{t-2}\binom{2^d-2}{t-1}) = 3.0$. We now need the following approximation.

Proposition 2. $N_{d,t} \leq \binom{2^d-2}{t-2}/(t-1) \leq (1+\epsilon_1)N_{d,t}$, *where* $\epsilon_1 = 0.00077, d \geq 17, 3 \leq t \leq 10$.

Proof. $N_{d,t} = (\binom{2^d-2}{t-2} - N_{d,t-1} - \frac{t-1}{t-2}(2^d-t+1)N_{d,t-2})/(t-1)$. As $N_{d,t-1}$, $N_{d,t-2} \geq 0$, $N_{d,t} \leq \binom{2^d-2}{t-2}/(t-1)$. Now $\frac{\binom{2^d-2}{t-2}/(t-1)}{N_{d,t}} = 1/(1 - \frac{N_{d,t-1}}{\binom{2^d-2}{t-2}} - (2^d - t+1)\frac{t-1}{t-2}\frac{N_{d,t-2}}{\binom{2^d-2}{t-2}})$. Again, $N_{d,t-1} \leq \binom{2^d-2}{t-3}/(t-2)$ and $N_{d,t-2} \leq \binom{2^d-2}{t-4}/(t-3)$. Substituting, $\frac{\binom{2^d-2}{t-2}/(t-1)}{N_{d,t}} \leq 1/(1-\frac{t-1}{t-2}\frac{\binom{2^d-2}{t-3}}{\binom{2^d-2}{t-2}} - (2^d-t+1)\frac{(t-1)^2}{(t-3)(t-2)}\frac{\binom{2^d-2}{t-4}}{\binom{2^d-2}{t-2}})$. Simplifying we get, $\frac{\binom{2^d-2}{t-2}/(t-1)}{N_{d,t}} \leq 1/(1-\frac{t-1}{2^d-t+1} - \frac{(t-1)^2}{2^d-t+2}) \leq 1/(1-\frac{t^2}{2^d-t+1})$. Hence $\binom{2^d-2}{t-2}/(t-1) \leq (1+\epsilon_1)N_{d,t}$, where, $\epsilon_1 = \epsilon + \epsilon^2 + \epsilon^3 + \ldots$ upto ∞, with $\epsilon = t^2/(2^d-t+1)$. It is clear that for a given t, as d increases ϵ_1 decreases. Fixing $d = 17$, for $3 \leq t \leq 10$, we have calculated the value of ϵ_1 and taken the maximum value, which is 0.00077. Hence the result. □

From Proposition 2, $s(\binom{c}{t-1}/(\binom{c}{t-2}\binom{2^d-2}{t-1}))(\binom{2^d-2}{t-2}/(t-1)) \leq 3.0 \times (1+\epsilon_1)$. Routine calculation shows that $s(\binom{c}{t-1}/(\binom{c}{t-2}\binom{2^d-2}{t-1}))(\binom{2^d-2}{t-2}/(t-1))$ equals $s\frac{c-t+2}{2^d-t}$. Thus, $s\frac{c-t+2}{2^d-t} \leq 3.0 \times (1+\epsilon_1)$.

Proposition 3. *Consider* $2^{\frac{d}{t-1}+\log_2(t-1)} \leq c \leq 2^{\frac{2d}{3}}$. *Then,* $\frac{c-t+2}{2^d-t} \leq \frac{c}{2^d} \leq (1+\epsilon_2)\frac{c-t+2}{2^d-t}$, *where* $\epsilon_2 = 0.316, d \geq 17, 3 \leq t \leq 10$.

Proof. $\frac{c}{2^d} - \frac{c-t+2}{2^d-t} = \frac{c2^d-ct-c2^d+t2^d-22^d}{2^d(2^d-t)} = \frac{(t-2)2^d-ct}{2^d(2^d-t)} \geq \frac{2^d-ct}{2^d(2^d-t)} \geq 0$, since for the range of c, t, d, the value $2^d - ct$ is always positive.

Again, $\frac{\frac{c}{2^d}}{\frac{c-t+2}{2^d-t}} = \frac{1-\frac{t}{2^d}}{1-\frac{t-2}{c}} \leq \frac{1}{1-\frac{t-2}{c}} = (1-\frac{t-2}{c})^{-1}$. So, $\frac{c}{2^d} \leq (1+\epsilon_2)\frac{c-t+2}{2^d-t}$, where $\epsilon_2 = \epsilon + \epsilon^2 + \ldots$ upto ∞, where, $\epsilon = \frac{t-2}{c}$. For minimum value of $c = 2^{\frac{d}{t-1}+\log_2(t-1)}$, where we take $d = 17, t = 10$, $\epsilon_2 = 0.316$. This is the case when ϵ_2 is maximum in the given range. Hence the proof. □

From Proposition 3, we get $s\frac{c}{2^d} \leq 3.0 \times (1+\epsilon_1) \times (1+\epsilon_2)$, which gives, $sc \leq 4 \times 2^d = 2^{d+2}$. From the above discussion, we get the following result.

Theorem 5. *Consider a primitive polynomial $f(x)$ of degree $d \geq 17$. Algorithm 2 needs s iteration to produce a t-nomial $(3 \leq t \leq 10)$ multiple having degree $\leq c$ of $f(x)$ with probability as high as 0.95, where $sc \leq 2^{d+2}$.*

Theorem 5 clearly identifies the time memory trade-off in the Algorithm 2. It is important to discuss the following issues in this direction. The best possible time memory trade-off is when $s = c = \lceil 2^{\frac{d}{2}+1} \rceil$. For cryptanalysis, one needs to find a sparse multiple at a degree as low as possible. To get a t-nomial multiple at a very low degree (i.e., low values of c), the number of iterations s will increase. In [5] it has been identified that given a primitive polynomial of degree d, the expected least degree of its t-nomial multiple is approximately $2^{\frac{d}{t-1}+\log_2(t-1)}$. Hence, it is expected that in Algorithm 2, minimum value of c should be taken as $2^{\frac{d}{t-1}+\log_2(t-1)}$. However, it is evident that in such a scenario the number of iterations s will be as high as $2^{d+2-\frac{d}{t-1}-\log_2(t-1)}$. In the next section, we will identify that the amount of memory in a computer (RAM) will mount an additional constraint on the value of c. Now we implement Algorithm 2 in a more specific form.

Algorithm 3 *Inputs as in Algorithm 1.*

```
1. Take array of integer $Arr[0, \ldots, c]$. Load $\alpha^i$ (d length bit patterns interpreted
   as integers) in $Arr[i]$, $0 \leq i \leq c$. Use another array of integer $Idx$ of length
   $c+1$ with $Idx[i] = i$. Sort Arr in ascending order and maintain corresponding
   order in Idx, i.e., after sorting, if $Arr[i] = \alpha^j$, then $Idx[i] = j$.
2. Take $(t-2)$ distinct integers $i_1, \ldots, i_{t-2}$ from $[1, \ldots, c]$ uniformly at random.
3. Calculate $\beta = 1 + \alpha^{i_1} + \ldots + \alpha^{i_{t-2}}$.
4. Use binary search to see if $\beta$ belongs to the array Arr.
5. If $\beta$ belongs to the array Arr, say $Arr[j] = \beta$, then $i_{t-1} = Idx[j]$. Report
   $1 + x^{i_1} + \ldots + x^{i_{t-2}} + x^{i_{t-1}}$ (it will either be a t-nomial or a $(t-2)$-nomial)
   and terminate. Else go to step 2.
```

Note that the space required for the algorithm is dominated by $2(c+1)$ integers needed for the arrays Arr, Idx (see step 1) in Algorithm 3. The time complexity for the sorting in step 1 of Algorithm 3 is $O(c \log_2 c)$. The expected number of iterations is s, where each iteration means execution of step 2 to step 5. Among these, step 5 needs $O(\log_2 c)$ time for binary search. Thus the time complexity is $O(c \log_2 c + s \log_2 c) = O((c+s) \log_2 c)$. From Theorem 5, let us consider the worst case, i.e., $cs = 2^{d+2}$. The time complexity is minimum when $c = s = 2^{\frac{d}{2}+1}$, i.e., the complexity is $O(d2^{\frac{d}{2}})$.

It is very clear that we are restricted in terms of available RAM in the computers. Let us explain it with an example. Currently a computer with 256 Megabytes (2^{28} bytes) is available at nominal cost. Consider that, we try to find sparse multiples of a degree 64 primitive polynomial. Now, storing each integer for the arrays Arr, Idx in step 1 of Algorithm 3 will need 64 bits, i.e., 8 byte space. Thus the maximum value of c is restricted by $2 \times c \times 8 = 2^{28}$, i.e., $c = 2^{24}$. Taking $c = 2^{24}$, the value of s comes around 2^{42}, which is computationally high. In fact, considering the memory requirements for the operating system and other parts of the program, the value of c will decrease further. If $c < s$, then the time complexity will be dominated by $s = 2^{\frac{d}{2}+a}$, where $a > 1$. In that case, the time complexity will be $O(s \log_2 c)$, i.e., $O(d2^{\frac{d}{2}+a})$.

To keep the value of s low, we need to go for higher c. This makes it clear that we are not in a position to use $2c$ amount of memory space as required in Algorithm 3. We like to point out that the existing sub exponential time algorithms for Discrete Log Problem (DLP) can be successfully used here [11].

Algorithm 4 *Inputs as in Algorithm 2.*

```
1. Take $(t-2)$ distinct integers $i_1, \ldots, i_{t-2}$ from $[1, \ldots, c]$ uniformly at random.
2. Calculate $\beta = 1 + \alpha^{i_1} + \ldots + \alpha^{i_{t-2}}$.
3. Use a routine to solve DLP and get $\beta = \alpha^{i_{t-1}}$.
4. If $i_{t-1} \leq c$, report $1 + x^{i_1} + \ldots + x^{i_{t-2}} + x^{i_{t-1}}$ (it will either be a t-nomial
   or a $(t-2)$-nomial) and terminate. Else go to step 1.
```

Note that step 3 of Algorithm 4 uses some routine to solve the DLP. One of the well known sub exponential algorithm due to Coppersmith is available at [11, Page 112]. Define $L_q[\alpha, \psi] = O(\exp((\psi + o(1))(\ln q)^{\alpha}(\ln \ln q)^{1-\alpha}))$, where ψ is a positive constant and $0 < \alpha < 1$. The expected run time of the Coppersmith's algorithm for $\mathbf{F}_{2^d}^*$ is $L_{2^d}[\frac{1}{3}, \psi]$ for $\psi < 1.587$. Algorithm 3 needs $O(\log c)$ time to get i_{t-1} in step 4. Using DLP in step 3 of Algorithm 4, this requirement is (expected) $L_{2^d}[\frac{1}{3}, \psi]$, where ($\psi < 1.587$), which is larger than $O(\log c)$. However, the advantage in Algorithm 4 is there is no constraint on the value of c as it relates memory requirement in Algorithm 3.

Now consider the case, where the polynomial is not primitive. Consider a polynomial $f(x)$ over GF(2) with degree d and exponent δ. We are interested in finding t-nomial multiples of $f(x)$ with degree $< \delta$. Consider the ring of residue classes $R = \frac{GF(2)[x]}{<f(x)>}$ whose elements are $g(x) + <f(x)>$, denoted by $[g(x)]$, with $g(x) \in GF(2)[x]$. From [9, Theorem 1.61, Page 25], it is clear that any element of R is linear combination of $1, x, x^2, \ldots, x^{d-1}$. The zero element of R is denoted by $[0]$. Consider the set G consisting of $[x]^i$, for $0 \leq i < \delta - 1$. Clearly $G \subset R$. As $e < 2^d - 1$, G does not contain all linear combinations of $1, x, x^2, \ldots, x^{d-1}$. It is important to note that G is a cyclic group with respect to multiplication modulo $f(x)$. Further G is not closed with respect to addition modulo $f(x)$. That is, it may very well happen that $[x]^i, [x]^j \in G$, but $[x]^i + [x]^j = [x^i + x^j] \notin G$, for some

$i, j \in \{0, 1, \ldots, \delta - 1\}$. Consider an expression $[x]^{i_1} + [x]^{i_2} + \ldots + [x]^{i_{t-1}} + 1$, for $\delta > i_1 > i_2 > \ldots > i_{t-1} \geq 1$. If this is equal to $[0]$, i.e., $[x^{i_1} + x^{i_2} + \ldots + x^{i_{t-1}} + 1] = [0]$, then $f(x)$ divides $x^{i_1} + x^{i_2} + \ldots + x^{i_{t-1}} + 1$. So we have a t-nomial multiple $x^{i_1} + x^{i_2} + \ldots + x^{i_{t-1}} + 1$ of $f(x)$.

So, we can apply the above mentioned algorithms, for getting t-nomial multiples of any arbitrary polynomial having exponent δ, considering the representations of $[x]^i$, for $0 \leq i < \delta$, as linear combination of $1, x, x^2, \ldots, x^{d-1}$ in $\frac{GF(2)[x]}{<f(x)>}$. Next refer to the algorithms presented in the last subsection. Consider a polynomial $f(x)$ having degree d and exponent δ. We choose $c < \delta$. Identify α as $[x]$, β as $[h(x)]$ and γ as $[g(x)]$. Here Arr contains the d bit representations of $[x]^i$. Observe that $[g(x)]$, $[h(x)]$ may not be in G, as G does not satisfy closure property with respect to addition modulo $f(x)$. In such case, t-nomial multiple will not be available and one has to go for next iteration.

References

1. A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in Cryptology - EUROCRYPT 2000*, number 1807 in LNCS, pages 573–588. Springer Verlag, 2000.
2. C. Ding, G. Xiao, and W. Shan. *The Stability Theory of Stream Ciphers.* Number 561 in Lecture Notes in Computer Science. Springer-Verlag, 1991.
3. S. W. Golomb. *Shift Register Sequences.* Aegean Park Press, 1982.
4. K. C. Gupta and S. Maitra. Primitive polynomials over GF(2) – A cryptologic approach. In *ICICS 2001*, number 2229 in LNCS, Pages 23–34, November 2001.
5. K. C. Gupta and S. Maitra. Multiples of primitive polynomials over GF(2). INDOCRYPT 2001, number 2247 in LNCS, Pages 62–72, December 2001.
6. K. Jambunathan. On choice of connection polynomials for LFSR based stream ciphers. INDOCRYPT 2000, number 1977 in LNCS, Pages 9–18, 2000.
7. G. A. Jones and J. M. Jones. Elementary Number Theory. Springer Verlag London Limited, 1998.
8. T. Johansson and F. Jonsson. Fast correlation attacks through reconstruction of linear polynomials. In *Advances in Cryptology - CRYPTO 2000*, number 1880 in Lecture Notes in Computer Science, pages 300–315. Springer Verlag, 2000.
9. R. Lidl and H. Niederreiter. Introduction to finite fields and their applications. Cambridge University Press, 1994.
10. F. J. MacWillams and N. J. A. Sloane. *The Theory of Error Correcting Codes.* North Holland, 1977.
11. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.
12. S. Maitra, K. C. Gupta and A. Venkateswarlu. Multiples of Primitive Polynomials and Their Products over GF(2). In *SAC 2002*, August 2002, pages 218–234 in preproceedings, proceedings to be published in Lecture Notes in Computer Science.
13. W. Meier and O. Stafflebach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1:159–176, 1989.
14. T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, IT-30(5):776–780, September 1984.
15. T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Computers*, C-34(1):81–85, January 1985.

A Secure Object Sharing Scheme for Java Card

Junqi Zhang, Vijay Varadharajan, and Yi Mu

Department of Computing, Macquarie University
Sydney, Australia
{janson,vijay,ymu}@ics.mq.edu.au

Abstract. Code and data sharing is an important issue in Java Card. The Java Card 2.2 introduced an object sharing scheme to allow client applets to access to the server applet methods using a sharable interface object. This method suffers from some security problems such as Applet Identifier impersonation. For solving those problems, we propose two secure object sharing schemes based on the delegate model introduced by Montgomery and Krishna. Our schemes have some significant improvement over the original model in that a client applet in our system needs only one key.

1 Introduction

Java Card can be referred to as a smart card that can run Java programs.[1,2] As showed in Figure 1, a Java Card contains an operation system, Java Card Virtual Machine (JCVM), Java Card Framework, and Industry add-on Classes in Read-Only Memory (ROM). Java Card Framework implements the Application Programming Interface (API). The applications are java packets that are located in Electrical Erasable Programmable Read Only Memory (EEROM). Every application consists of one or more applets. Firewall partitions the Java Card platform object system into separates protected object spaces called context. All applets within same Java packet share the same context. The Java Card Runtime Environment (JCRE) has its own JCRE context, and this context has special system privileges and can perform operations.

Some existing secure issues in Java Card 2.1 (same in Java Card 2.2) have been addressed by Montgomery and Krishna[3]. In order to solve the secure problems, they introduced a delegate model where each of applets (client or server) has a delegate. To invoke a method in the server applet For each sharable, a client shares a secret key with the server; therefore the client can be authenticated by the server with a simple challenge-response protocol. It is noted that in their scheme the number of secret keys for a client applet is proportional to the number of sharable methods in the server applet. In this paper, based on the delegate concept, we introduce two alternative object sharing schemes for Java Card 2.2[4]. Our schemes are also based on a distributed encryption that allows an encryption key to map multiple decryption keys. Using this property, in our system each client applet needs only one secret key, which shows a significant improvement over the previously proposed model.

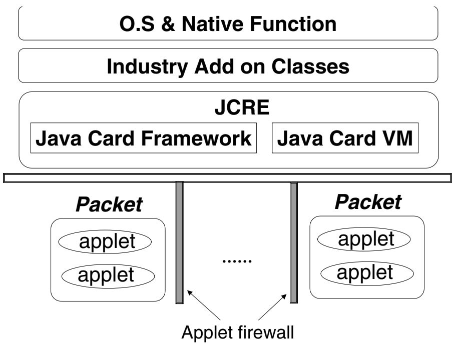


Fig. 1. Java Card architecture.

The rest of this paper is organized as follows. In Section 2, we introduce the object sharing mechanism in Java Card 2.2 and discuss some major security issues in Java Card 2.2. In Section 3, we present two new object sharing schemes. In Section 4, we conclude the paper.

2 Object Sharing Mechanism in Java Card 2.2

Java Card 2.2 specification[5] introduces four mechanisms to enable applets to interact with each other and with the JCRE, so one context can access an object belonging to another context. These mechanisms are JCRE Entry point Objects, Global Arrays, JCRE Privileges and Sharable Interfaces. Here we discuss the Sharable Interface.

2.1 Applet Isolation

Applet isolation means that one applet can not access the objects or fields in another context unless the other applets provide an interface for access. For security concern, except the Java technology protection, Java Card uses the Applet Firewall to enforce the VM, and allows the VM to automatically perform additional security checks at runtime. This is because the applet entry points are public; other applets may obtain an object reference to access the applet and might cause the sensitive data to be leaked. The only method the JCRE provided to access other applets is through the SIO mechanism.

2.2 The Applet Context

The Java Card platform object system is partitioned into separate protected object space by the firewalls. These partitions are referred to as contexts. As shown in Figure 2, all applets in the same package share the same context. There is no firewall between the applets within the same package. The JCRE has its own context and has special privileges to perform operations that are denied to the context of applets. There is only one active context within the Virtual Machine (VM). The VM determines if the context switch is required.

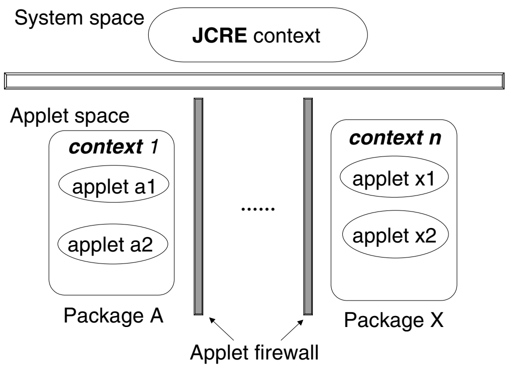


Fig. 2. Contexts with the Java Card Platform's Object System.

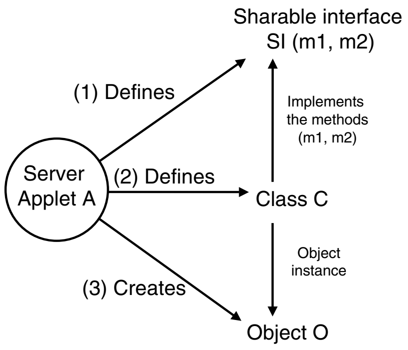


Fig. 3. Server Applet create Sharable Interface Object (SIO).

2.3 Sharable Interfaces and Sharable Interface Object (SIO)

A sharable interface was introduced in the Java card 2.1 API[6]. It defines a set of methods that are available to other applets. A class can implement any number of sharable interfaces and can extend other classes that implement sharable interfaces. A sharable interface object is an object class that implements a sharable interface. Only the methods defined in sharable interface can be accessed by applets in other contexts, other methods and fields in SIO only can be accessed in the same context. The applets that provide the SIO are server applets and the applets that use the SIO are client applets. The applet can be a server to some applets and a client to other applets.

2.4 Creating a SIO

Figure 3 illustrates how to create a SIO. To create a sharable interface object, the server applet first defines a sharable interface SI that includes the method the server applet supplied to other client applets and then defines a service provider class C that can implement the methods defined in the SI. After that, the server

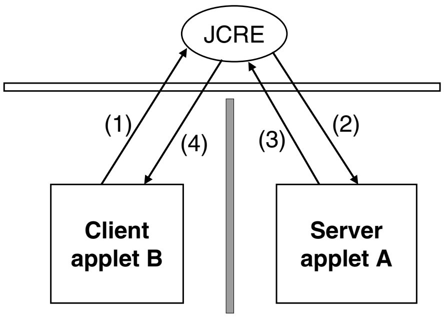


Fig. 4. The client applet requests the sever SIO. (1) Request a SIO, Jcsystem.getAppletShareable InterfaceObject (Server AID,byte), (2) Receive AID of B, getAppletShareable InterfaceObject (Client AID,byte), (3) Return reference of O or null and (4) Receive reference of O or null and store it in object reference variable SIO.

applet creates an object of the service provider class C. When the applet instance is created, it is registered in the JCRE using the Applet Identifier (AID). During the object sharing, this AID is used to identify the applet.

2.5 Requesting and Using a SIO

The Figure 4 shows the process that the client applet requests the server SIO.

The client applet can invoke the any methods from the sharable interface. During the invocation the Java Card VM performs a context switch, the context of the server applet becomes the currently active. At this time, the firewall allows the client applet to access all the methods and fields in the SIO, and prevents it from accessing all other methods and fields in non-shared object in the server applet context. The context of the client applet is not visible at all, and the firewall prevents the server applet to access the context of the client context.

2.6 The Issues with the SIO

There are some security issues in Java Card 2.1/2.2 having been addressed.[7]. We summarise the important issues as follows:

- The JCRE security protection mechanisms do not prevent interfaces from being cast into other kinds of interfaces that might exist for given object. If one client applet is granted to use any sharable interface, it can cast into other interface. So it is possible to for this client applet to access all of the sharable interface methods of an object.
- In the sharable interface mechanism, the sever applet provides the sharable interface to some particular applets, which solely depends on the AID of the clients applet. In other words, the server determines whether to provide the service to a particular client applet, just checking the AID of the client applet. In this case, a rogue applet may maliciously set the AID to be the

same AID of a client applet, which is known to have access to the server applet particular sharable interface, and then request the service from the server. The server applet only has the AID of the client for reference and this request AID match it, so it will grant this request. Then the rogue applet can freely access the server sharable interface for any purpose.

– It is hard for future reference to a shared object. In this mechanism the server applet that has sharable interfaces must have a list of AID of client that have granted to access the sharable interface. This is not a problem for the new written and late loaded applet that legitimately needs to access the server applet sharable interface. This is because the server applet grants the access only based on the AID of the client applet, and the AID of such applets is excluded from this server list. The server applet must be rewritten and reissued if the client applet have to share the interface.

3 New Object Sharing Schemes

In this section, we propose two object sharing schemes. We refer to them as Scheme 1 and Scheme 2. Scheme 1 is simple, but it satisfies the important authentication requirements for Java Card. Scheme 2 gives an alterative for authentication, but it has some additional features that the privilege can be granted and removed from a client applet. The most important improvement of these systems over the existing one is that a client applet needs only one secret key.

3.1 Construction of Keys

Our model is a variant of the scheme given in Ref. [8]. Basically, we want to create a “public key” system such that several private keys are mapping to a single “public key”. By “public key” we actually mean “encryption key”. We should avoid using the term “public key” because all keys in our system are private!

Let p be a large prime, $\mathbb{Z}_p^*$ be a multiplicative group of order $q = p - 1$, and $g \in_R \mathbb{Z}_p^*$ be a generator. To construct an encryption key for Method j associated with n decryption keys, we need to construct a the polynomial function of order n, $f(x) = \prod_{i=1}^{n}(x - x_i) \bmod q$, here $(x_1, ..., x_n)$ are decryption keys. Clearly, if we set $f(x) = 0$, then $x_1, ..., x_n$ are solutions to the equation. We can write the polynomial function into a general form: $f(x) = \sum_{i=0}^{n} a_i x^i \bmod q$. The unique encryption key can be then constructed as an $(n+1)$-tuple $(g^{a_0}, g^{a_1}, ..., g^{a_n})$. For simplicity, we denote by $(g_0, g_1, ..., g_n)$ the encryption key tuple.

To encrypt a message $M \in \mathbb{Z}_p$, we select a number $r \in_R \mathbb{Z}_q$ set $u_i = g_i^r$ for $i = 1, ..., n$, and compute $C = m(g_0)^r$. The ciphertext consists of $(u_1, ..., u_n, C)$. To decrypt the message, you must have one of decryption keys associated with the public key. For x_j, you can decrypt the message as follows:

$$Cu_1^{x_j} u_2^{x_j^2} \cdots u_n^{x_j^n} = Mg^{r \sum_{i=0}^{n} a_i x_j^i} = M$$

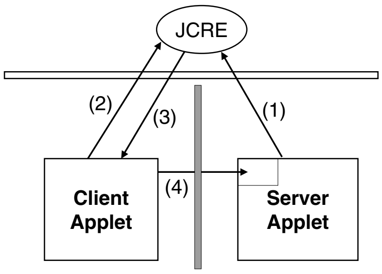


Fig. 5. Invoking a method in the server applet.

3.2 The Java Card Object Sharing Model

Let us have a quick review of object sharing in Ref. [3]. The simplified illustration is given in Figure 5 and described as follows:

– The server applet has the sharable methods. All the sharable methods must register in the JCRE system using the method and AID as parameter, `register(Method, Server AID)`. This is referred to as Step 1.
– In Step 2, the client applet obtains access to the server applet via JCRE using `JCSystem.getDelegate(Server AID, byte)`
– In Step 3, the server applet delegate object is sent to the client.
– In Step 4, the client accesses the shared method in the server applet. Prior to this step, there should be an authentication process that will be given in the main protocols later.

Note that the JCRE has been changed by adding two system methods: `register(Method, Server AID)` and `JCSystem.getDelegate(Server AID, byte)`.

3.3 Scheme 1

For convenience in our presentation, we denote by $y(x_1, ..., x_n)$ an encryption key wrt decryption keys $x_1, ..., x_n$. Assume that $M_1, ..., M_l \in \mathcal{M}$ are sharable methods in the server applet. Each of these methods is assigned with a private decryption key, x_j. On the other hand, each of client applets is assigned with a unique private encryption key, $y_j(.)$, which can be referred to as a certificate. In general, a client applet could have the right to access to several methods. For example, $y_j(x_2, x_4)$ represents the encryption key associated with methods M_2, M_4. In other words, Client j can have access to M_2 and M_4 with its private encryption key $y_j(x_2, x_4)$.

Following Figure 6, the protocol is described as follows (assume that the client has an index j):

1: CD $\rightarrow$ SD: The Request for service (this can be done by invoking the methods in the server)

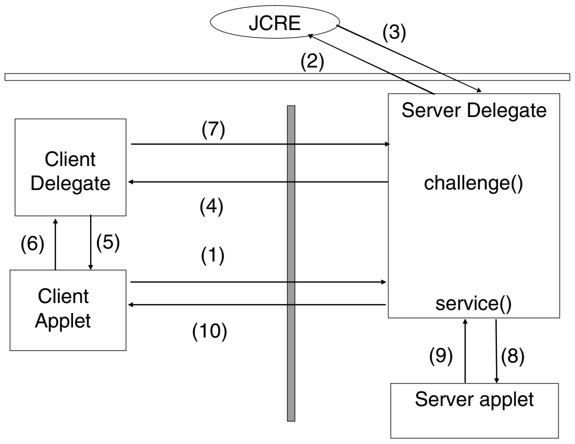


Fig. 6. Applet delegate approach to object sharing.

2: SD $\rightarrow$ JCRE: The Request for the client applet's AID. The server delegate has to determine the security lever of protecting the sharing method. If the method or the data is public to every context, the SD will provide the service to the client straightaway. If it is a sharable method, the client must be validated before provideing the service.
3: JCRE $\rightarrow$ SD: AID of CD
4: SD $\rightarrow$ CD: a random Challenge, c
5: CD $\rightarrow$ CA: The random Challenge c
6: CA $\rightarrow$ CD: The Response encrypted with y_j. Note that CA has a unique encryption key that is also served as its certificate. As an example, we assume that $y(x_1, x_5)$ is the encryption key and the CD wants to invoke Method M_5.
7: CD $\rightarrow$ SD: The Response encrypted with y_j.
8: SD $\rightarrow$ SA: The Response encrypted with y_j and the Service request by the CD (i.e., M_5 in the example). SA then checks whether the response can be decrypted using x_5. A positive result indicates that the CA has the privilege to access the method (M_5).
9: SA $\rightarrow$ SD: The Response for the Service (if the CD Response can be decrypted using proper decryption keys associated with the requested methods)
10: SD $\rightarrow$ CA: The Response for the Service. SD can now invoke the method (M_5).

3.4 Scheme 2

We now give the second scheme that is opposite to Scheme 1. Each method in SD is assigned with an encryption key and each of client applets is assigned with a unique decryption key, x_i. In general, a client applet could have the privilege to access several methods, so we denote by $y_i(.)$ the encryption key associated with method i. For example, $y_2(x_2, x_4)$ is the public key for Method 2, so only Client Applet 2 and Client Applet 4 can invoke the method.

The encryption keys of all methods are stored in a special directory in JCRE that is only readable to the server applets. These encryption keys can be updated by an access control software called AC in JCRE. Updating happens when a client applet leaves or joins the Java card and when the access right of a client applet needs to update.

Following Figure 6, we give the protocol below:

1: CA $\rightarrow$ SD: The Request for service (we assume that Client Applet 5 is the CA and that Method M_3 is intended to invoke.)
2: SD $\rightarrow$ JCRE: The Request for the AID of the client applet, CD, and for the encryption key related to the shared method requested by the client (noting that the server delegate has to determine the security level the shared methods. If the method is public to every context, the CD will provide the service to the client straightaway.)
3: JCRE $\rightarrow$ SD: AID and the encryption key (say, $y_3(x_3, x_5, x_6)$)
4: SD $\rightarrow$ CD: a random Challenge, c, encrypted using the matching encryption key (i.e., $y_3(x_3, x_5, x_6)$ in the example)
5: CD $\rightarrow$ CA: The encrypted Challenge, c. CA decrypts it using its decryption key (x_5).
6: CA $\rightarrow$ CD: The Response wrt c (this can be done after the decryption of the challenge, c, is successful.)
7: CD $\rightarrow$ SD: The Response wrt c. SD checks whether or not c matches the one sent to CD in Step 4.
8: SD $\rightarrow$ SA: The Service request by the CD.
9: SA $\rightarrow$ SD: The Response for the Service request.
10: SD $\rightarrow$ CA: The Response for the Service request. CA is now able to invoke M_3.

In comparison with Scheme 1, Scheme 2 has an additional feature for revocation. A client applet's access rights can be changed by modifying the encryption keys stored in the special directory in JCRE without changing the private decryption key of the client applet.

4 Conclusion

We have proposed two schemes for securing object sharing in Java Card, using the concept of Delegate. Our schemes eliminates the security problems mentioned in Section 2; therefore only legitimate client applets can have access to the specific sharable methods. Our schemes are based on a distributed asymmetric key system where an encryption key maps to multiple decryption keys. This property naturally fits into the Java Card environment such that in our system a client applet needs only a single key exhibiting significant improvement over the previously proposed system.

References

1. "Java card special interest group.," http://www.javacard.org/others/what-is-java-card.html.
2. "Java card applet developer's guide," tech. rep., July 17, 1998.
3. M. Montgomery and K. Krishna, "Secure object sharing in java card," in *USENIX Workshop on Smart card Technology*, May 10-11 1999.
4. "Java card 2.2 virtual machine specification," tech. rep., 2002. //java.sun.com/products/javacard/JCVMSpec.pdf.
5. "Java card 2.2 paltform specification," tech. rep., 2002. //java.sun.com/products/javacard/.
6. "Java card 2.2 application programming interfaces specification," tech. rep., 2002. //java.sun.com/products/javacard.
7. "Java card 2.2 runtime environment specification," tech. rep., 2002. //java.sun.com/products/javacard/JCRESpec.pdf.
8. Y.Mu, V. Varadharajan, and K. Q. Nguyen, "Delegated decryption," in *Cryptography and Code, Lecture Notes in Computer Science*, Springer Verlag, 1999.

IDS Interoperability and Correlation Using IDMEF and Commodity Systems

Nathan Carey, Andrew Clark, and George Mohay

Information Security Research Centre
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434,Brisbane
Queensland, 4001
{n.carey,a.clark,g.mohay}@qut.edu.au

Abstract. Over the past decade Intrusion Detection Systems (IDS) have been steadily improving their efficiency and effectiveness in detecting attacks. This is particularly true with signature-based IDS due to progress in attack analysis and attack signature specification. At the same time system complexity, overall numbers of bugs and security vulnerabilities have increased. This has led to the recognition that in order to operate over the entire attack space, multiple IDS must be used, which need to interoperate with one another, and possibly also with other components of system security. This paper describes an experiment in IDS interoperation using the Intrusion Detection Message Exchange Format for the purpose of correlation analysis and in order to identify and address the problems associated with the effective use and management of multiple IDS. A study of the process of intrusion analysis demonstrates the benefits of multi-IDS interoperation and cooperation, as well as the significant benefits provided by alert analysis using a central relational database.

Keywords: Intrusion Detection, Data Analysis, Correlation, Interoperability, Network Management

1 Introduction

Intrusion Detection Systems (IDS) have evolved significantly over the past two decades since their inception in the early eighties [6]. The simple IDS of those early days were based either upon the use of simple rule-based logic to detect very specific patterns of intrusive behaviour or relied upon historical activity profiles to confirm legitimate behaviour. In contrast, we now have IDS which use data mining and machine learning techniques to automatically discover what constitutes intrusive behaviour and quite sophisticated attack specification languages which allow for the identification of more generalised attack patterns.

IDS are still however commonly characterised according to a two-fold taxonomy involving detection method on the one hand and placement on the other. Classification by detection method relates to signature-based vs. anomaly-based

IDS while classification by placement relates to host vs. network based IDS. Many IDS can be described in this way, with the exception of emergent hybrid IDS, which may use multiple placement types or detection methods.

Signature based IDS constrain the range of attacks detected using specific detection patterns in return for acceptable detection error rates, while anomaly based IDS cover the entire attack space by looking for anomalies within their data source at the cost of increased error rates.

Host based IDS or HIDS, use host logs and host event records to provide a record of current activity which can then be analysed. Network based IDS or NIDS, typically use packet headers and packet level information, sometimes even packet payload, as their working data. Due to the speeds required in modern networks, NIDS are typically signature based.

With a wide range of IDS and applications, it is inevitable that individual IDS have their own areas of specialisation and effectiveness [7]. As a result, with the broad range of systems and networks in use today, many systems use multiple IDS which brings with it the associated challenge of achieving a consistent approach to IDS management and analysis of IDS alerts.

The Intrusion Detection Working Group (IDWG) emerged following previous work done by the Common Intrusion Detection Framework (CIDF), both efforts attempting to standardise and formalise the work of cooperating IDS. The IDWG efforts so far have focussed on two standards, a data exchange format and a protocol for communication. The Intrusion Detection Message Exchange Format (IDMEF) built on the experience of CIDF, but given the increased deployment of XML for specifying protocols across the Internet used XML for expressing the two protocol requirements. The IDMEF Data Type Definition (DTD) is currently in version 1.0, and provides a rich and extensible alert representation for a broad range of IDS applications.

The motivation and background for our research is described in more detail in Section 2. Section 3 describes the detailed design of the prototype software that has been developed for interoperability and alert correlation, as well as testing procedures and experiments performed with the prototype. Section 4 then presents a case study of an experimental attack analysis, while Section 5 addresses future work and conclusions.

2 Related Work and Motivation

Our work is related to work by Vigna et. al.[10], Doyle et al. [5], Valdes and Skinner [9], Cuppens [1] and Debar and Wespi [4]. In each case, use is made of a central alert store that captures alerts from multiple sensors in order to assist the overall intrusion detection process. Some of the systems use IDMEF for communication, and some utilise purpose built IDS platforms, rather than relying on commodity IDS for alert data. While all perform some sort of analysis on the data, the mechanisms used are different. The STAT framework [10] provides the ability to perform dynamic configurability of STAT components and uses Java to aid in portability. The MAITA [5] project reflects similar goals, but utilises a more complex architecture to support interoperability and uses trend templates as opposed to STATL (used by the STAT framework) for the specification of chains of events. Valdes and Skinner use a probabilistic approach to perform correlation of information from multiple sensors, and focus on the concept of 'threads' to maintain links between alerts. Debar and Wespi use features in the Tivoli Enterprise Console to perform correlation, and focus on the abilities of an management system to reduce the amount of data presented to an administrator. Recent work by Cuppens [1,2] which focuses on commodity IDS, and uses a central database for alert aggregation and analysis is the most similar to our own approach. Cuppens uses a Prolog database and static signatures for attack detection, together with stored procedures to perform aggregation of alerts to reduce redundancy. The signature set is defined using the LAMBDA syntax, which enables the specification of very complex event relationships. Our work, by comparison focuses on a simple framework built of commodity and free software in order to produce practical alert correlation across heterogenous commodity IDS. Our work also differs in concepts to do with implementation, such not using stored procedures, the nature of interaction between analysis and signatures, and the format and implementation of signatures themselves.

We use the following terms and definitions inherited from both recent work cited above, but we define them again here for convenience.

Attack: A series of steps taken by an attacker to achieve an unauthorized result; any attempt to gain knowledge of or penetrate a system; includes scanning, probing, mapping, etc.[1]

Alert: An alert is a warning message generated by an IDS. It may indicate an attack or suspicious event.

Complex Attack: An attack that may comprise multiple steps and generates multiple alerts. This definition is based on that by Cuppens [1].

Attack Signature: An attack signature is the specification of a pattern of alerts whose occurrence would indicate a complex or multi-step attack. In our work, signatures consist of a combination of groups and/or sequences of alerts, in any order.

Alert Cluster: An alert cluster is a group of alerts that are related by one or more common features, such as time, source or destination. This relationship may be logical, mathematical or based on statistical grouping. We use this definition of cluster as opposed to some others e.g., in the work of Cuppens, who uses 'cluster' to term grouped alerts corresponding to a single instance of an attack.

Our work has been motivated by the previous work above and attempts to address the application of IDS data analysis within a simple architecture based upon commodity IDS. As a result, our goals are:

– to provide standardised IDS interoperability;
– to provide a capability for multiple levels of analysis of multi-IDS alerts;
– to provide centralised management of IDS.

Interoperability is important in order to provide a proper base for information sharing amongst IDS and possibly other system security components e.g., be-

[1] http://www.cert.org/security-improvement/modules/m06.html

tween IDS and firewalls, or host logging information. This allows us to exploit the use of heterogeneous IDS and other components to provide identification and notification of a wider range of alerts than is possible with homogeneous IDS.

The need for alert analysis arises from the two main tasks performed by an IDS administrator - to identify individual attacks and group them by characteristics, and the ability to investigate a specific stored alert to discover other alerts that may be related.

While the first task is simple data aggregation and representation, the second relies on correlation, the selection of sets of alerts that could be related, and induction by ascertaining which alerts in the set of relevant alerts should be investigated further.

Centralised management of IDS enables us to integrate the features of interoperability and global IDS data analysis (in this case correlation) into a product that is both useful and efficient. While interoperability and correlation can be achieved without requiring a central management framework, centralised management facilitates system extensibility with regard to incorporation of new tools, and efficiency with regard to removal of redundant or superfluous functionality.

In particular correlation provides us with the potential ability to see beyond the actual alerts themselves, and determine trends, find patterns and infer relationships between alerts - capabilities which a single sensor is unlikely to have, and can be far more easily and comprehensively performed with multiple IDS.

3 Prototype for IDS Interoperability

One of the goals of our research has been to produce a prototype that draws upon standardised communication and alerting formats, as well as making use of commodity systems. Our approach is to convert native alerts from Snort and Dragon to the IDMEF format, store these IDMEF alerts in a central database, and perform analysis on the data both with native SQL queries and custom algorithms. We feel this commodity IDS solution suits the 'real world' concerns of many IDS users, rather than being customised for particular IDS, which ties usage to particular products. The decision to use common IDS unfortunately means that some aspects of an alert may be ignored in favour of maintaining a common 'core' set of information provided by the majority of IDS. The exact ways in which this effects our prototype are discussed later in the paper.

There are a selection of free tools for IDS analysis, dealing almost exclusively with Snort alerts. Examples include ACID, Demarc and SnortReport. The aim of our prototype is to provide the functionality of these free tools as well as more sophisticated event processing and more detailed correlation capabilities. Some systems, including some commercial offerings, provide administrators with a web browser interface to manage their systems. While this may be appropriate for some systems, the range of features we desired required a full GUI, with the ability to run on multiple OS. Two IDS were chosen for the prototype , due to their suitability both in terms of OS platforms and alert format. These two IDS were Snort and Enterasys Dragon.

After first looking at the possibilities of the native Snort alert format for simple centralised IDS alert storage (due to the tools currently available for Snort mentioned above), it was decided, for interoperability reasons, to look at the capabilities of the emerging IDMEF standard.

While IDMEF is a very expressive and extensible format, our usage is limited to the common set of information available from virtually any IDS, that of alert classification (or alert name), time, source and target. Omitted information that could be considered useful with more widespread IDS support includes priority of the alert, host names and DNS names of hosts, and the impact of the alert.

3.1 The Architecture

The prototype architecture consists of three major components: the Alert Agent (AA), the Control Unit (CU) and the Administrative Console (AC). These three components interact in a manner so as to provide a simple architecture and allow for the majority of applications to be integrated easily and effectively.

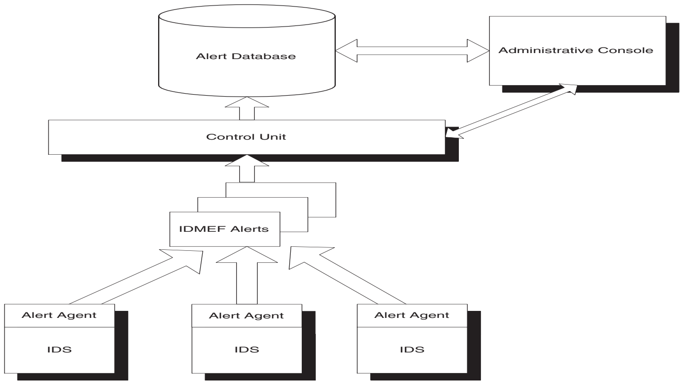


Fig. 1. The Architecture

Alert Agent. The Alert Agent performs translation and communication of IDMEF alerts to the CU. It can utilise either a native IDS IDMEF output format such as the Snort output plugin, or translate to IDMEF from the native IDS format. Generally 'native' (IDS generated) IDMEF messages should be considered a more accurate data source than converted messages, due to the increased amount of information that can be gathered at alert time versus conversion later, but the basic information required in an IDMEF message can be met by virtually any commonly used IDS alert format.

Control Unit. The recipient of all IDMEF messages is the Control Unit. The CU is responsible for alert processing and storage, as well as implementing any real-time alert analysis (not implemented at this time). When an alert is received it is queued for storage in the database. This queue is buffered which allows our prototype to handle influxes of alerts, such as in the case of a DoS attack. When storing, the alert is converted to the SQL tables by a custom IDMEF-DBMS mapping module and a JDBC connection pool. This allows IDMEF alerts to be stored in a normal relational database, in this case a Postgres database, aiding in portability and the effectiveness of later searches on alert data. The database contains all the information stored in the IDMEF alert, though only a small amount of this is generally used in our analysis. With the addition of more support from IDS vendors to supply the additional information possible in an IDMEF alert, analysis could include alert priority and impacts in order to perform better tracing of possible cause and effect relationships between alerts.

Administrative Console. This application allows the administrator to gain access to the alerts stored in the database. In the current off-line method, the simplest form of correlation, the aggregation and sorting of alerts, is performed by SQL queries on the database. Complex correlation (where mathematical patterns maybe involved) is performed by Java algorithms working on the produced database result set. The process by which data is analysed is discussed further below. The Administrative Console is designed to be the only device actually reading the database, though there could be multiple consoles running simultaneously. The console also has rights to alter information inside the database if required, to perform tasks such as delete old records, merge alert information or change alert data to add extra information.

3.2 Data Analysis

Figure 2 shows the four step process we use to analyse IDS data and shows the waterfall nature of the stages and the information flow between them. This model builds on concepts mentioned in related work discussed above. We use this model to systematically reduce the workload and volume of information in later stages, and to concentrate the amount of relevant data to be used in the correlation and induction stages. The basic process of alert analysis uses the stages below to group alerts into clusters which are related in some way, either by an attack signature or by a pattern, such as time difference or grouping by time. How these relations are developed is described below.

Stage 1: Data Aggregation. At this point, alerts are grouped by time, classification, source and destination using similarity metrics. These metrics currently use simple factors such as time period, time difference between alerts (to detect sequences), IP differences and matches on alert classification, host or IDS type.

Stage 2: Data Reduction. After determining which alerts can be grouped together, we remove redundant alerts. This can be performed by merging alert groups based on pre-defined criteria (such as ignoring certain types of messages

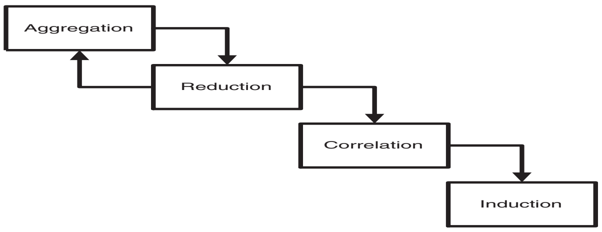


Fig. 2. Data Flow Diagram

from a particular IDS) and deleting the originating alerts, or using signatures for patterns of alerts to delete. Examples of this could be low-level alerts of no interest, alerts not appropriate for the environment, such as IIS attacks on an Apache web server, or duplicate alerts. The goal is to remove that information that is definitely not relevant, therefore while the process of removing alerts is simple, the logic can become quite complex. Reduction in this iterative stage fashion is not currently used in our system, though the major functionality is implemented off-line using SQL.

Stage 3: Data Correlation. With the alerts reduced to a more relevant subset, the software considers which alerts may be correlated. Correlation is currently based on time, IP, location, and name, but in principle could be virtually any field deemed relevant in determining an attack. Examples of this could be noticing sequences in fields like ports (such as in the case of a portscan), or even the process of an attack on a HIDS (such as might occur in a resource exhaustion attack). The relationship may be based directly on a signature, or by the system watching for certain abnormal operation, such as the above cases of sequences of ports opened by a single host (for a portscan) or a large amount of processes opened by a single user (for the above host attack). The exact nature of what the system can determine as 'abnormal operation' is yet to be defined.

Stage 4. Data Induction. At this point in analysis, the software will extrapolate from the data either to determine events that may have occurred, or would be likely to occur. Induction uses more complex analysis than does correlation but the same general principles apply. The major difference is that induction is used to extrapolate or interpolate information from the dataset, rather than simply determine relationships. Examples of this could be predicting the next stage of an attack, deducing missing components of an attack, or determining possible spread of an attack through the network. In our current prototype, induction is not yet included, though will be closely tied to the 'abnormal operation' aspect of correlation.

3.3 Attack Signatures

Attack specification languages such as LAMBDA [3], and CISL [8] provide the ability to define very complex relationships between *events* themselves. However, our only requirement at this stage is that the signatures be able to describe the relationships between the *alerts* generated by IDS as opposed to events. In order to best solve our specific problems, we have developed our own signature format for patterns of alerts. Our signature scheme is designed to be simple yet still have the capability to define reasonably complex patterns. It is used only to represent alert relationships, and as such requires less complexity and expressiveness than the above systems. The general requirements for our signatures are:

- The signature must contain information about the conditions on the alerts comprising the signature. This includes whether the alerts are a sequence or unordered group, any temporal constraints on the interval between alerts, any spatial constraints (i.e., same host or subnet) and the timeout for expiry of the signature.
- Signatures should be able to generate an alert themselves or be able to trigger new monitoring processes, so multiple signatures can be combined in a sequence or hierarchy if necessary.

A signature may be of two types:

1. For signatures which define alert sequences, the monitor process attempts to find a match for the first item only, and when triggered by such a match, a new monitor process is spawned which looks for the next step in the sequence. This means a sequence of *n* alerts could potentially have *n-1* monitor points. A sequence signature monitor expires when the interval between individual steps is reached, or the overall signature timeout is reached.
2. For signatures which define alert groups, the monitor must check for each item in the group. For this reason, groups should be kept small, or significant processing could be required. When an alert within the set is matched, it is flagged so that it will not be matched against again. A group signature monitor expires when a timeout occurs on the interval between alerts. A signature-wide timeout is not needed, as functionally, this would be equivalent to the timeout between alerts.

One property of our system, is that signatures can specify what operations to perform on the alerts which contributed to the signature (e.g. store, delete, merge), as well as any responses to be effected. This relatively simple system specification can accommodate many conceivable relationships of interest by being able to express signature composition using multiple signatures, without storing redundant information. An example of our signatures being used to describe a multi-step attack across two different IDS in multiple stages is included in the Evaluation and Testing section.

We currently use a first-fit system to determine matching on signature monitors, which means that the first signature which matches an alert will prevent any other signatures from matching it. This has implications for signature ordering and the construction of signatures, but is not a significant impediment at this stage of research. Investigation into other methods for matching, such as 'best-fit' or 'multi-fit'approaches may be used later if needed.

3.4 Discussion

The architecture is purposely minimalistic. Other approaches propose more sophisticated architectures which are likely to be difficult to manage and maintain. We have purposely not included a comprehensive communication mechanism or information sharing protocols. As our system is a testbed for data analysis, more complex issues of distributed monitoring, information sharing policy and complex data channels would distract from the focus. We consider our simple communication to be an adequate model in many cases, especially in small to medium sized implementations, or those already using secure communication channels (such as a separate ethernet network dedicated to IDS) or IDWG's communication protocol, the Intrusion Detection eXchange Protocol (IDXP).

The current system is implemented in an off-line context, in order to properly evaluate the capabilities of the system for centralised monitoring and analysis without the distraction of requiring real-time operation. We have nonetheless endeavoured to include consideration of performance and eventual real-time implementation while developing the architecture.

4 Evaluation and Testing

In order to properly gauge the usefulness of our system, we experimented with various attacks and profiled the operation of our system on a set of attack data versus the operation required by a human administrator. This helped us distinguish those elements of the system that were useful, and identified algorithms used in human analysis that could be replicated in our prototype.

4.1 Testing

The testing procedure was developed both to test the operation of the system itself, and to evaluate the success of the methodology for data analysis. This entailed the careful construction of a test network to enable the evaluation of analysis across network boundaries as well as investigation of the abilities of one IDS to reinforce or invalidate the data of another. Figure 3 shows the testing framework for the software. Two 'client' machines are used, each hosting Snort, Dragon Squire and Dragon NIDS. The alerts from these systems are then interpreted by IDS Alert Agents, and sent to the Control Unit on a separate third machine. The Control Unit is co-located with a Postgres Alert Database, and stores all alerts received in the database. This is then accessed from a separate machine acting as the Administrative Console. All the machines were placed on a switch, to separate traffic into distinct network segments.

A suite of attacks capable of being launched across multiple machines was selected, consisting of two DDoS tools, stick and tfn2k (Case 2 below), as well as multiple variations of nmap scan (Case 1 below) and a multi-stage attack described by Cuppens [3] (Case 3 below). A vulnerability scanner and two attack tools were also run, but not described below as the alerts could not be correlated

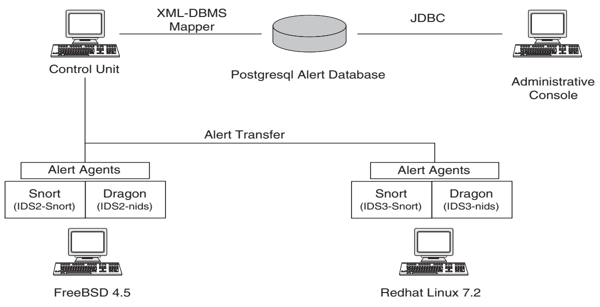


Fig. 3. Testing Framework

directly with the attacks themselves. The Cuppens attack consists of 6 steps, of which only five (all except step 5) are detectable by Snort and Dragon:

1. finger root@target
2. rcpinfo <target>
3. showmount <target>
4. mount <target directory>
5. cat "++" < .rhost
6. rlogin <target>

The attacks were scripted in order to satisfy reproducibility concerns, and the session was logged in TCPDUMP for later analysis. Initial experiments had used TCPreplay and TCPDUMP to reproduce attack sets, but the sophistication of Dragon and Snort meant that some data 'normal' replayed (such as obsolete TCP SYN/ACK sequences) would trigger alerts, and the incorrect session generation meant that some attacks were (correctly) ignored as erroneous. Because of these problems, we used shell scripting to automate the attacks. Using scripting meant that the attacks could be performed quickly and repeatably for multiple iterations if required and avoid the problems of stale packets with TCPDUMP.

4.2 Correlation Analysis

After running the three attacks (Case 1, Case 2 and Case 3) below, we commenced correlation by querying how many alerts had been logged. The four IDS, IDS2-dragon, IDS3-dragon ,IDS2-snort and IDS3-snort produced a total of 8799 alerts. A list of alerts per IDS shows IDS2-dragon had 7 alerts, while IDS2-snort had 3737, IDS3-dragon 3784 and IDS3-snort 1271.

Case 1: TCP Scan. Looking at the abnormally low number of alerts on IDS2-dragon, we observed that it contained three unique types of alert, one 'HEARTBEAT', one 'SSH:VERSION-1', and five 'TCP-SCAN'. The TCP-Scan alerts are potentially dangerous, and have timestamps of 1:22:52, 1:22:55, 1:23:10, 1:23:14 and 1:24:18. Based on the similarity in times we can infer three separate events have occurred. By searching through the database for time periods +/- 5 seconds around these groups of alerts on other IDS, we observed that many scan-related alerts (such as 'ICMP Ping NMAP' and 'Scan Proxy' messages from Snort and a 'TCP-SCAN' message from Dragon) occurred across both IDS2 and IDS3 in the 10 second periods around 1:22:50 and 1:23:10, and that the third time (1:24:18) is the result of separate traffic. Considering there were 18 alerts (including a normal HEARTBEAT message from IDS3-Dragon) from the four different IDS (2 IDS2-Dragon, 11 IDS2-Snort, 3 IDS3-Dragon, 3 IDS3-Snort) in the first period, and 11 alerts (2 IDS2-Dragon, 7 IDS2-Snort, 1 IDS3-Dragon, 1 IDS3-Snort) in the second, we can safely conclude that two separate large-scale scans occurred.

Case 2: DoS Attack. As noted above, the other alerts around 1:24:18 alert did not correspond to a TCP-Scan. From 1:24:18 to 1:24:49 3551 alerts from varying IP's logged as 'BAD-TRAFFIC' requests on IDS2-Snort. From 1:24:38 to 1:24:48 1246 'BAD-TRAFFIC' alerts were received on IDS3-Snort. Interestingly, IDS3-Dragon logs 3758 alerts in this time, indicating that Dragon on FreeBSD may contain DoS protection in the kernel network code or the FreeBSD version of Dragon itself. Large amounts of varying source alerts could indicate either a spoofed single host DoS attack, or a bone-fide multi-host DoS attack. As there were 1625 unique IP's, all of which had the last IP quad as '0', one could reasonably assume a spoofed DoS attack has taken place.

Case 3: Cuppens Attack. Without extra events to analyse, by looking at the relative levels of alerts on both IDS, we note that IDS3 has more alerts than IDS2, and that Snort logs less alerts than Dragon. The large amounts of alerts for this host indicate a higher chance of attack, therefore warrants further investigation. In much the same way as Case 1 above, we can look at the types of alerts on IDS3 and relate these to each other in order to identify attacks.

Using the smaller dataset on IDS3, and using a query to show a list of unique alert names, Snort logs 9 different types of alert: 2 types of 'BAD_TRAFFIC', plus 'FINGER root query', 'ICMP PING NMAP', 'RPC portmap listing', 'RPC portmap request mountd', 'RSERVICES rlogin root', 'SCAN Proxy (8080) attempt' and 'X11 outbound client connection detected'. Looking at the occurrences of each of these, 1260 come from BAD TRAFFIC alerts (related to the DoS attack, above), two occurrences of the SCAN Proxy alert, two occurrences of ICMP PING NMAP, three occurrences of RPC portmap listing and one of FINGER root query, RPC portmap request mountd and RSERVICES rlogin root. Looking at the times of the alerts, 1:23:46 contains two instances of RPC portmap listing, as well as the RPC mountd and RSERVICES Login attempt, comes 30 seconds before the DoS attack, and contains the majority of types of alerts the IDS logged - this makes the alerts from this particular time suspicious.

A search for alerts from all IDS for 10 seconds around this time gives us 16 entries confined to IDS3, 5 from Snort and 11 from Dragon. This period adds the "FINGER root query" from Snort mentioned earlier, plus a FINGER:ROOT, RSH:ROOT and 8 DYNAMIC-TCP messages from Dragon. The FINGER and

RSH:ROOT messages match those of Snort, while the DYNAMIC-TCP messages match the RPC messages of Snort. These alerts constitute a match to the sequence that indicate Cuppens multi-stage attack. In this case Dragon's alerts allow us to validate the results of Snort. Note that both systems missed the 'cat "++"' stage - it is at this point the addition of a host-based IDS would help to increase detection scope and effectiveness. Notice that this attack can be identified from others simply by looking for anomalous alerts and correlating information between IDS.

This attack can also be detected using signatures. In our case, we note that "FINGER root" and "FINGER:ROOT" are equivalent, so create a signature of the name "DUAL IDS Finger Root Detection" consisting of both of these alerts, and we create another with "RSH:ROOT" and "RSERVICES rlogin root" called "DUAL IDS Remote Root Login Detection". If we combine these in a sequence with the RPC messages from Snort, as well as a single occurrence of "DYNAMIC-TCP" (to cover the Dragon-Snort RPC overlap) we create a signature for the combination of these alerts that allows the checking of detection across heterogenous IDS. This allows for better error rates due to the reduced likelihood of false positives across both IDS simultaneously.

Summary. The end result is that we are able to identify signatures of certain large-scale attacks across time and space, as well as the anomalies which may indicate suspicious activity with simple SQL queries and a small amount of extra analysis. We replicated the experiment of Cuppens [1] "multi-stage" attack using Snort and Dragon instead of Snort and E-trust, without complicated attack signatures.

We see that the levels of data reduction and aggregation are included in the SQL query generation, while alert correlation occurs within an outside process. The current system iterates through the data either in the manner shown above looking for abnormal events, or by searching for matches to simple static signatures. It cannot currently extrapolate or interpolate information, but that functionality is the subject of further research.

As well as demonstrating the process of an attack analysis, this example also shows the advantages of cross-IDS and cross-locational analysis. In all cases, the overlap of IDS alerting provided an increase in information regarding the nature of the attack, and correlating across location helped to identify the scale of the attack.

5 Conclusions

Our aim in developing the prototype has been to implement alert correlation across multiple real-world IDS. We have achieved successful development of such a prototype to support IDS interoperability using several concepts that build upon previous work, as well as new concepts, such as simple signature specification and grouping of alerts. This has demonstrated proof of concept of the ideas used in the research and it now remains to deploy the prototype in a wider range of real-world situations, in particular the context of a wider range of attacks and background traffic, in order to better explore the architecture's potential for analysis using the concepts of aggregation, reduction, correlation and induction. Further work will address optimisation of the algorithms used in the architecture, and deployment of the system in a real-time environment.

References

1. Frédéric Cuppens. Managing alerts in a multi-intrusion detection environment. In *Annual Computer Security Applications Conference (ACSAC 2001)*, Sheraton New Orleans, Louisiana, USA, December 10-14 2001. ACM.
2. Frédéric Cuppens and Alexandre Miége. Alert correlation in a cooperative intrusion detection framework. In *2002 IEEE Symposium on Security and Privacy (S&P '02)*, pages 187–202, Berkeley, CA, USA, May 12 - 15 2002.
3. Frédéric Cuppens and Rodolphe Ortalo. Lambda: A language to model a database for detection of attacks. In *Third International Workshop on Recent Advances in Intrusion Detection, (RAID 2000)*, volume 1907 of *LNCS*, pages 197–216, Toulouse, France, October 2-4 2000. Springer.
4. Herve Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *Proceedings of the 4^{th} International Symposiun on Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 85–103, Davis, CA, October 2001. Springer-Verlag.
5. Jon Doyle, Isaac Kohane, William Long, Howard Shrobe, and Peter Szolovits. Agile monitoring for cyber defense. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX-II)*, Anaheim, California, June 12-14 2001. IEEE.
6. John McHugh. Intrusion and intrusion detection. *International Journal of Information Security*, 1:14–35, 2001.
7. Marcus Ranum. Coverage in intrusion detection systems. Technical report, NFR Security, Inc, 26th June 2001.
8. Brian Tung. The common intrusion specification language: A retrospective. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX), 2000*, pages 3–11, Hilton Head, SC, 2000. IEEE.
9. Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *Proceedings of the 4^{th} International Symposiun on Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 54–68, Davis, CA, October 2001. Springer-Verlag.
10. G. Vigna, R.A. Kemmerer, and P. Blix. Designing a Web of Highly-Configurable Intrusion Detection Sensors. In *Proceedings of the 4^{th} International Symposiun on Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212 of *LNCS*, pages 69–84, Davis, CA, October 2001. Springer-Verlag.

A Synthetic Fraud Data Generation Methodology

Emilie Lundin, Håkan Kvarnström[*], and Erland Jonsson

Department of Computer Engineering
Chalmers University of Technology
412 96 Göteborg, Sweden
{emilie,erland.jonsson}@ce.chalmers.se
hakan.k.kvarnstrom@telia.se

Abstract. In many cases synthetic data is more suitable than authentic data for the testing and training of fraud detection systems. At the same time synthetic data suffers from some drawbacks originating from the fact that it is indeed synthetic and may not have the realism of authentic data. In order to counter this disadvantage, we have developed a method for generating synthetic data that is derived from authentic data. We identify the important characteristics of authentic data and the frauds we want to detect and generate synthetic data with these properties.

Keywords: fraud detection, synthetic test data, data generation methodology, user simulation, system simulation

1 Introduction

Fraud detection is the process of automated analysis of data from a service with the goal of revealing attempts to use the service without paying or in some other way illicitly benefit from the service. When designing a fraud detection system, it is essential to have suitable data for evaluation and testing. This data must be representative of normal and attack behavior in the target system since detection systems can be very sensitive to variations in input data.

Using synthetic data for evaluation, training and testing gives several advantages compared to using authentic data. Data properties of synthetic data can be tailored to meet various conditions not available in authentic data sets. On the other hand purely synthetic data suffers from the fact that they may be quite unrealistic and will consequently not properly reflect the properties of authentic data. Thus, the aim of this work has been to develop a methodology for generating synthetic data with realistic properties and this is achieved by using authentic data as a property "seed". Thus, the method uses statistical properties from smaller amounts of authentic data and generates large amounts of synthetic data preserving parameters important for fraud detection and the training of fraud detection systems. Examples of such parameters are user and service behavior.

[*] The author is also with Telia Research AB, SE-123 86 Farsta, Sweden

The rest of this paper is organized as follows. Related work is summarized and discussed in section 2. Advantages and motivation for using synthetic data can be found in section 3 and our method for generation of synthetic data is presented in section 4. Conclusions can be found in section 5.

2 Related Work

There are very few papers which focus on properties of test data. Most papers describing training and test data focus on testing a specific detection prototype, or is about benchmarking and comparing detection systems. Often, data is only mentioned briefly. In most papers, (manipulated) authentic data is used.

The absence of research on synthetic data for fraud detection applications has led us to study work in the intrusion detection area. In [10] we discuss several similarities between fraud and intrusion detection systems. However, there are some differences that need to be considered. Intrusion detection is performed on log data from operating systems, applications, and networks to detect malicious actions in a computer system. Fraud detection is normally performed on log data from a specific service to find people trying to gain unauthorized benefits from the service. Fraud detection is often more specialized and adapted to a service as opposed to intrusion detection which is more general. Roughly, fraud detection can be seen as application-specific intrusion detection. We believe that the way test and training data is retrieved and used is similar and the same methods for generating data can be used.

In the following two sections, we review some papers that present test methods using authentic and synthetic test data. We describe how the data is retrieved, used, and manipulated, and how it compares to our method.

2.1 Authentic Data

In the JAM project [2], some papers have been written that actually focus on test and training data properties. For example, [4], and [15] describe experiments on varying the fraud rate in training data and also introduces a cost model where each transaction is associated with a cost. In [15] which is an unpublished technical report, the data and the manipulation of data is described more thoroughly, but this information is shortened in the published papers based on the same experiments. For the experiments two large sets (500000 records each) of authentic credit card transactions are used. The transactions are labeled as fraudulent or legitimate and the fraud rate is around 20%. A cooperation with two banks provided the data. The labeling seems to have been performed by bank employees and the authors mention that some of the unclear cases were indeed labeled as fraudulent. The authors concluded that the optimal rate of fraudulent vs. normal events in training data depends on the cost model and the learning algorithm used. For their applications, the desired rate is shown to be close to 50:50. The fraud categories presented in their work are likely to be representative of future fraud. However, the process for labeling data is not described.

In [3], data is described but not analyzed in much detail. The data used for the experiments in this paper is GSM Toll Tickets from Vodafone. It consists of calls made by 300 normal users during two months. This data is "sanitized", but it is not specified how this is done. It also consists of calls made by 300 fraudulent users but it is not explained how these fraudulent users are selected. The paper describe what information the Toll Tickets contain although the ratio between normal and fraudulent events is probably not realistic.

2.2 Synthetic Data

Probably the biggest effort in testing and comparing intrusion detection systems is made by DARPA in 1998 and 1999 [11]. A great deal of work has been spent on generating large amounts of test and training data for this project. The generation of data is best described in [8]. The test data contains network traffic and system call log files (SUN BSM log files) from a simulated large computer network. Both attacks and background data has been generated synthetically, but the background data is said to be similar to sampling data from a number of Air Force bases. Background data is generated mainly by using software automata simulating the usage of different services. Attack data is generated by running attack scripts. The database of attacks used is described in [9]. Some more complicated background data and attacks are injected live into the system. No extensive analysis of the quality of the generated data has been made. Some efforts in automating the data generation further is described in [7].

McHugh [13] criticizes the lack of validation of test data in the 1998 DARPA evaluation. The process of generating data is vaguely described, and it is difficult to determine the quality of the data. There are no guarantees that the attacks used are really representative and realistic. Also, he questions whether the background data is really representative of the user behavior in the computer systems used and if the behavior in these systems represent user behavior in other computer systems. Another question he raises is whether attacks are realistically distributed in the background data. Some improvements were made in the 1999 experiments but many of these issues are still not fully addressed.

Debar et al. [6] developed a generic intrusion detection testbed. They show how to simulate user behavior with a finite state automata to obtain normal traffic, but state that this is only a practical approach if the set of user commands is limited. Instead, they use recorded live data from user sessions, which has been replayed inside the workbench. They also describe how they create attack scripts to use in the testing process.

Maxion and Tan [12] discuss the effects of more or less irregular background data. They generate "random" background data with different degrees of regularity and show that it affects the false alarm rate drastically. The paper also measures regularity in real-world log data.

Chung et al. [5] claim that concurrent and distributed intrusions are likely to occur in a computer system, and therefore, should be included in test data for intrusion detection systems. They have developed a tool that parallelizes attack scripts to simulate concurrent intrusions.

Puketza et al. [14] describe a software platform for testing intrusion detection systems. They have used the UNIX package *expect* to generate synthetic user sessions. However, they do not analyze the quality of this data to any great extent.

From the discussion above we conclude that most projects seem to use synthetic data due to lack of authentic data or lack of data having desired properties. In the DARPA project [1], it was necessary to generate synthetic data since it was not possible to retrieve the desired amount of data. Also, most projects seem to suffer from a shortage of real attacks, which makes it necessary to inject fraud/intrusions synthetically.

3 Using Synthetic Data for Evaluation

Training and testing of fraud detection systems require data which has certain properties. Access to authentic data from real services is often preferred although it may suffer from lack of control of what fraud cases it contains and the amount of data may not be sufficient. Another possibility is to use synthetic data.

Synthetic data can be defined as data that is generated by simulated users in a simulated system, performing simulated actions. This definition can be relaxed to include humans performing simulated actions on a system. Simulated actions means that people (or a program) perform actions according to a specification created by the experiment organizers, and not act according to their normal behavior. This specification should reflect the desired behavior of the system. In this paper, we use the wider definition of synthetic data. It depends on the situation whether it is better to simulate user behavior using a software automata or to hire people to generate background data and attacks. There is also a choice between using a fully simulated system, a real system or a mix of real and simulated system components.

3.1 Rationale

Use of authentic data is not always a viable solution for evaluation of detection systems. For future services (e.g. services that are planned or under development) authentic data may not exist or may only be available in small quantities. In these situations, synthetic data is the only possible solution for conducting tests. Moreover, expected fraud cases for services under development are not present in the authentic data as the service has no real users. Under these circumstances, synthetic data containing expected fraudulent user behavior must be generated for testing of the detection system.

An advantage of synthetic data is that it can be designed to demonstrate properties, or include attacks, not available in the authentic data. This gives a high degree of freedom during testing and training. In addition, synthetic data may be generated to cover extensive periods of time which could have taken months or years to collect from the real target system. Also, a large number of users can be efficiently simulated, even though the real system only has a few of them. This makes scalability tests possible.

3.2 Data for Training Fraud Detection Systems

The process of detecting fraud involves analyzing large amounts of data, looking for signs of fraudulent behavior. Intelligent techniques such as neural networks and other AI (Artificial Intelligence) techniques can be used to decide whether an event or a group of events indicate fraudulent behavior. Common to most intelligent techniques is that extensive training needs to be performed. During training, data similar to that of the target system must be available. In addition, the data must be labeled (i.e. all events must be categorized as normal or fraudulent) so that the detection algorithm can learn to distinguish normal usage from fraudulent. We have identified a number of properties that characterize good training data.

- Data is labeled, i.e. we have exact knowledge of which attacks are included in the data.
- The attacks in the input data represent the attacks we expect to occur in the target system. (Not necessarily the same attacks that currently occur in the system.)
- The number and distribution of attacks in the background data (fraud/ normal data ratio) are adapted to the detection mechanism. Some detection methods perform better if they are trained on data where attacks are over-represented. In [4], it is shown that varying the amount of attacks in the data will greatly affect the training process of the detection algorithms.
- The amount of data is large enough. In particular, certain AI algorithms need huge amounts of training data.

These properties indicate that synthetic data can be a better choice for training fraud detection systems.

3.3 Data for Testing Fraud Detection Systems

The detection algorithms need to be tested on data sets containing expected fraud in the system. Testing the detection capability of the algorithm may require a different data set than used during training, even though most of the properties for training data is valid also for test data. For example, it is easier to test the system if data is labeled. Some additional important properties for test data are stated here.

- The number and distribution of attacks in the background data (fraud/ normal data ratio) should be realistic.
- The attacks in the input data are realistically integrated in the background data. For example, time stamp of an attack, time between attacks, and time between parts of an attack may affect detection results.
- Normal (background) data should have similar statistical properties as authentic data from the target system. Different behavior in the system may affect detection performance drastically. In [12], it is shown that the false alarm rate rises sharply when background data becomes more irregular. This would indicate that test data is often system specific.

Testing a fraud detection system involves many diverse activities. Scalability tests need to be performed to see whether the systems can handle current and future data volumes. Again, large authentic data sets may not be available to realistically conduct such tests. Various stress tests can be conducted to test the system's ability to withstand sudden changes in data volume or the characteristics of the data. For example, the system can be tested to determine its susceptibility to parameter changes. These changes could be triggered by sudden changes of the environment in which the detection system operates or by faults and errors in the collected data. By using synthetic data, faults and other unexpected events can be injected to study the susceptibility of the detection system and its ability operate under harsh conditions.

4 Data Generation Methodology

There is a great deal of complexity in synthetic data generation and it is time-consuming to create the necessary components. Therefore, a methodology is needed to structure the work and to point out the choices that have to be made.

The main components needed to automate the data generation process are specifications of desired user behavior in the system, a user/attacker simulator, and a system simulator. The goal of the methodology is to guide the production of these components. The starting-point of the methodology is the collection of information about the anticipated behavior in the target system. The methodology includes both background and attack data generation. Therefore, we need both information about possible attacks as well as normal usage. This data serves as basis for user and system modeling.

4.1 Methodology Overview

In figure 1, the methodology for generating the data generation components is pictured. The first step is the *collection of data* that should be representative of the anticipated behavior in the target system. The data may consist of authentic background data, background data from similar systems, authentic attacks, and other collections of possible attacks. The second step is to *analyze the collected data* and identify important properties such as user classes, statistics of usage, attack characteristics, and statistics of system behavior. In step 3, the information from the previous step is used to identify parameters that need to be preserved to be able to detect the anticipated attacks, and to *create user and attacker profiles* that conforms to the parameter statistics. A *user model* is created in step 4. This model must be sophisticated enough to preserve the selected profile parameters. Also, the attacks are modeled in this step. The user and attacks simulators implements the models. In step 5, the system is modeled. The model must be accurate enough to produce log data of the same type as the target system for the input user actions. The system simulator is implemented according to this model. It is important that it is possible to configure the user and system models using variable input parameters in order to change the properties of the generated data during operation.

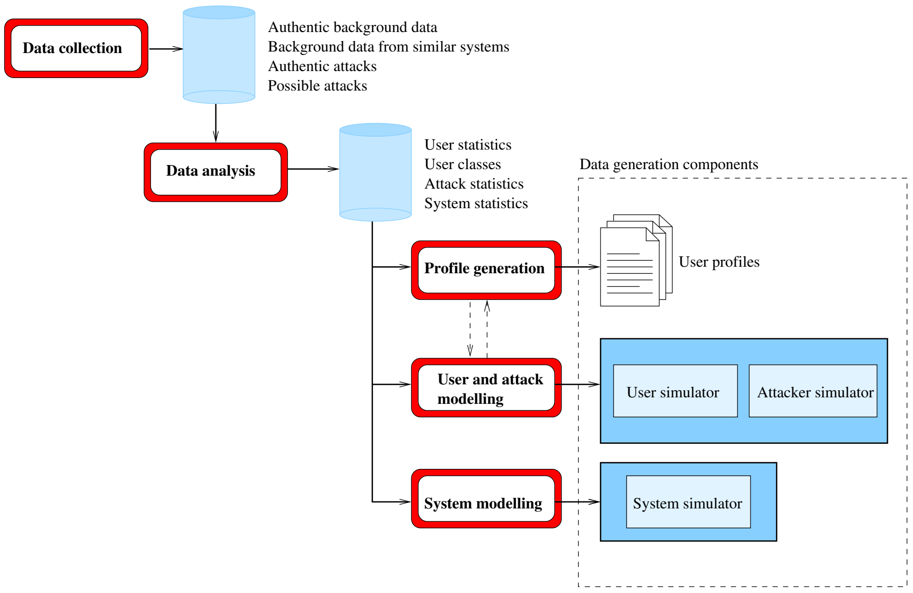


Fig. 1. Synthetic log data generation method

In the following sections, each of the steps in the data methodology are discussed and methods for implementing them are suggested.

The division into steps with well defined interfaces reduces the complexity of the task and makes it possible for different groups of people to work on different tasks. It is possible to run some of the work in parallel. For example, as soon as some authentic background data is collected and analyzed, it is possible to start working on both the user model and the system model. In parallel to the user and system modeling, attack data may be collected and analyzed to create the attack model.

It is possible to use people instead of a user simulator to create the user actions. It is also possible to use the whole, or parts of the real system instead of a system simulator. In some situations, this may be preferable, especially if the system or user behavior is very complex, and needs to be modeled in great detail. However, there are some disadvantages in using people and real hardware and software in the generation process.

If only simulation programs are used, it is possible to make the data generation process fully automatic. In the DARPA evaluation [8], they use humans to inject attacks, and a great deal of system hardware and software. They admit that their data generation procedure required much manual administration and attention during the simulation runs.

An advantage of using fully simulated components, is that it has the possibility to become very scalable, both concerning the number of users and the simulation time period. If humans are used to generate background data, each person can only perform the tasks of one or a few simulated users. If we want to use humans to inject attacks, the simulation time can not be much faster than real-time. Also, if real software and hardware is used, these components may limit the simulation speed considerably.

The rest of this section presents and discusses the different steps of the methodology in some detail and describes the final creation of the synthetic log data.

4.2 Data Collection

The data we need as a starting point in the data generation method are samples of background data and attacks representative of the anticipated behavior on the target system. The output log data from the target system is the input to the detection system. This is the type of data that should be produced in the synthetic data generation process. Therefore, it is convenient to have samples available of authentic data from the target system. Hopefully, this data also contains information about user and system behavior which is representative. This may be the most valuable source of data for the generation. If too small amounts of authentic data is available, it is also possible to collect data from similar services. In each case, we need to determine if this data is applicable to our service. Even if we have authentic data, these may not be representative, e.g. because the number of users are going to increase or functions in the system are changed before the detection system will be in operation.

Authentic attacks are often not available. Therefore, we need to collect them in other ways. In the intrusion detection area, databases of known attacks are more or less publicly available. In the fraud area, they are often more service specific, and we may need to "invent" possible attack scenarios or adapt known frauds from other types of services to our situation. Collected attacks may be injected into the target system to get corresponding log data. It is important that the log data can be labeled to know exactly which entries corresponds to a specific attack.

It is difficult to know that properties of the collected data corresponds to the environment that the detection system will operate in. On the other hand, this is not a problem specific for synthetic data. If the detection system is trained and tested on data with the "wrong" properties, the detection will not be correct irrespective of the type of input data. We must predict the future behavior of users and hope that the prediction is close enough to reality.

The output from this step may have many different formats. Some of it may be labeled log data on the same format as will be used as input to the FDS. It may also be databases of attacks with only some information about each attack.

4.3 Data Analysis

The next step is to analyze the collected data. The goal of this work is to get a picture of how the system is used and how user actions and attacks show up in log data.

One task is to identify classes of users with similar behavior. Exploratory Data Analysis (EDA) [16] is an existing set of ideas on how to study data sets to uncover the underlying structure, find important variables, detect anomalies etc. We believe that techniques based on those ideas could be used for this purpose. This may include use of visualization tools and/or clustering methods. It is important to use several classes of user behavior to get diversity in the generated data.

Another task is to identify important attack features, i.e. parameters that are useful for detection of the anticipated attacks. For example, to detect an attacker guessing passwords on a system, the number of failed logins within a certain time interval is a useful parameter. Also, system statistics must be examined. For example, response time and amount of network traffic generated by different events may be useful for the system modeling.

The log data generated by the target system should be examined to determine if it is adequate for effective detection. If it is not adequate, it may be necessary to implement additional logging mechanisms and collect new data.

The output from this step is user classes and a number of statistics for different aspects of user, attacker, and system characteristics.

4.4 Profile Generation

The next step is to identify important parameters for user behavior in the statistics. One way to identify these parameters is to study the features needed to detect expected frauds. These features must have correct statistical properties in the generated data to be useful for detection. One example of a fraud indicator would be users logging in at night that usually only log in during daytime. In this case, we must preserve the statistical distribution of logins during the day for the simulated users. Also, correlation between parameters may be fraud indicators and must be preserved.

When we know what parameters we have and what properties we want to preserve, collected data is used to find statistical values for them. There are more or less automatic tools available for calculating values for parameters and fitting data to statistical distributions.

Output from this step is files for different user classes containing values for all parameters that are required for the user simulation. A cooperation between this step and the user modeling step is needed to adapt the format of the profiles to fit the user model. For example, there are different ways to model the frequency of a specific event. Either, it can be defined as the expected number of events per time interval, or it can be defined as the expected time until the next event.

4.5 Modeling the User

The user and attacker behavior can be very complex to model. To limit the complexity, we should bear in mind that it is only of interest to simulate actions that will affect log data. It is in general easier to model the behavior of users in a service which is less complex. Behavior profiles are appropriate to use as configuration data to the user simulator. This way, we can easily adapt the simulator to changes in behavior and add new user classes.

There are several ways to model user behavior. One way is to use a finite state machine. This is suggested by Debar et al. [6] and also used in the DARPA evaluation [11]. This is a suitable method if user behavior is not too complex or if only a limited number of parameters need to be preserved in the background data. Also, it is possible to generate great amounts of data for many simulated users. However, it may be very time consuming to model users and attacks if the behavior needs to be modeled in great detail.

Another way is to "record" real user sessions and attack sessions and replay them in the system simulator. This method is suggested by Puketza et al. [14]. If the behavior is complex, this may be a better method than modeling the users with a state machine. The problems are that the system simulator must be advanced enough to handle this type of input and it may be difficult to generate large enough amounts of user sessions.

A third way is to use test suites developed for testing functionality in operating systems. This is suggested by Debar et al. [6]. The test suites are not representative of real user behavior but generate all sorts of events that may be very infrequent in a real-life system. The advantage of this method is that it does not require much work to implement, and it may be useful for evaluating the false alarm rate when behavior is very irregular. However, it is questionable if this type of data is useful for testing fraud detection systems.

The output of this step is "lists" of user actions in a format suitable to use as input to the system simulator.

4.6 Modeling the System

The simulated system must be able to produce output data similar to that of the target system, where the similarity can be restricted to those features that are necessary for the fraud detection. The simplest way to model the service realistically would be to set up a replica of all system components. This would give us perfectly accurate log data as long as the "users" are behaving realistically. However, it may require a great deal of resources if we want to simulate a large system, and there are other disadvantages as mentioned before.

A more realistic way to model the system would be to make a hardware replica of the central system components, but implement the clients in software. One single machine can simulate numerous clients. This is the method used by DARPA [11]. There are two major problems with this solution. Firstly, it is not trivial to simulate traffic from many different IP addresses from one machine, but it can be done. Secondly, it is not possible to speed up the simulation time as the client software must send out the traffic in real-time if the rest of the components are to respond realistically.

A third solution would be to implement a model of the service in software. Some of the real service software can be used, but we can experience problems if we want to run the simulation at non real-time speed. The advantages of this solution are that we have full control of the simulation process and can speed up the simulation as fast as the computing platform allows. Furthermore, simulation processes can be quickly and easily implemented for low complexity services. The disadvantage is that it may take some time and effort to study the behavior of the service to be able to implement the required functionality. There is also a risk of oversimplification resulting in significant differences between synthetic and authentic data.

The output of this step is data on the same format as the data produced by the target system.

4.7 Creation of Synthetic Log Data

The basic data generation methodology was presented in figure 1. Figure 2 below shows in more detail how the data generation components interact in the generation process.

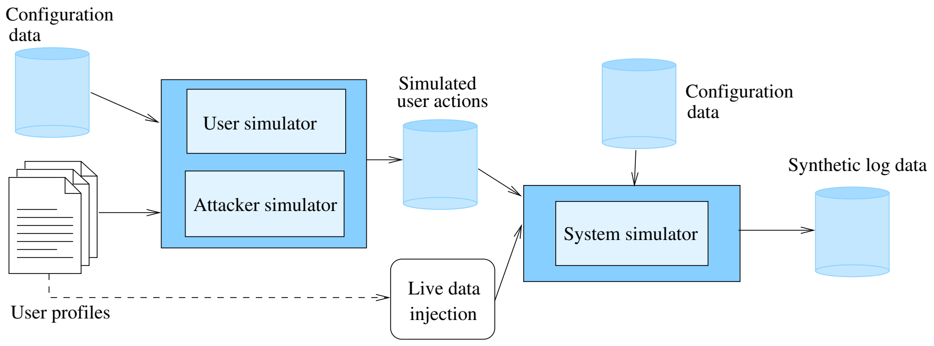


Fig. 2. The synthetic log data generation process

The *user profiles* are used as input to the *user and attacker simulator*. The *user and attacker simulator* generates user actions that are fed to the *system simulator*. The *system simulator* creates the final synthetic test data and the *configuration data for the user and attacker simulator* contains information that controls the generation of normal user and attack actions. For example, it may be the start time and stop time for simulation, the number of normal users from different user classes, the number of attacks of each type, and the start time and stop time for different attacks. The *configuration data for the system simulator* contains for example information about the number of clients that should be simulated, statistics about reply times and traffic amounts for different events, and behavior in the presence of certain attack events. Parameters that we want to vary between different simulation runs should be included in the configuration data. This means that we can generate batches of synthetic data for different purposes without reprogramming the simulators.

It is possible to complement the user simulation with *live data injection*, e.g. to generate more complex series of actions or to generate deviating behavior for special stress tests.

5 Conclusions

The results presented in this paper form a foundation for generation of synthetic test data based on authentic data for fraud detection systems. By starting out from small sets of authentic log data, appropriate synthetic log data can be created in large amounts. Properties of parameters in the initial data are preserved or can be tailored to meet the needs of testing and training of fraud detection systems. In the near future we aim to use our method in practice with authentic log data collected from a pilot-test video-on-demand application.

References

1. DARPA Intrusion Detection Evaluation. http://www.ll.mit.edu/IST/ideval/ The main web page for the DARPA evaluation experiments. July 2001.
2. JAM project homepage. http://www.cs.columbia.edu/ sal/JAM/PROJECT/ July 2001.
3. Peter Burge, John Shawe-Taylor, Yves Moreau, Bart Preneel, Christof Stoermann, and Chris Cooke. Fraud Detection and Management in Mobile Telecommunications Networks. In *Proceedings of the European Conference on Security and Detection ECOS 97*, pages 91–96, London, April 28-30 1997
4. Philip K. Chan, Wei Fan, Andreas L. Prodromidis, and Salvatore J. Stolfo. Distributed Data Mining in Credit Card Fraud Detection. *IEEE Intelligent Systems*, 14(6), Nov/Dec 1999.
5. Mandy Chung, Nicholas J. Puketza, Ronald A. Olsson, Biswanath Mukherjee. Simulating Concurrent Intrusions for Testing Intrusion Detection Systems: Parallelizing Intrusions. In *Proceedings of the 1995 National Information Systems Security Conference*, pages 173-183. Baltimore, Maryland, October 10-13 1995.
6. H. Debar, M. Dacier, A. Wespi, and S. Lampart. An Experimentation Workbench for Intrusion Detection Systems. Technical Report RZ2998, IBM Research Division, Zurich Research Laboratory, Zurich, Switzerland, March 1998.
7. Joshua Haines, Lee Rossey, Rich Lippmann, and Robert Cunnigham. Extending the 1999 Evaluation. In *Proceedings of DISCEX 2001*, Anaheim, CA, June 11-12 2001.
8. Joshua W. Haines, Richard P. Lippmann, David J. Fried, Eushiuan Tran, Steve Boswell, and Marc A. Zissman. 1999 DARPA Intrusion Detection System Evaluation: Design and Procedures. Technical Report 1062, MIT Lincoln Laboratory, February 2001.
9. Kristopher Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, MIT, 1999.
10. Håkan Kvarnström, Emilie Lundin, and Erland Jonsson. Combining fraud and intrusion detection - meeting new requirements. In *Proceedings of the fifth Nordic Workshop on Secure IT systems (NordSec2000)*, Reykjavik, Iceland, October 12-13 2000.

11. Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579-595, October 2000. Elsevier Science B.V.
12. Roy A. Maxion, and Kymie M.C. Tan. Benchmarking Anomaly-Based Detection Systems. In International Conference on Dependable Systems and Networks, pages 623-630, New York, New York, June 2000. IEEE Computer Society Press.
13. John McHugh. The 1998 Lincoln Laboratory IDS Evaluation: A Critique. In *Recent Advances in Intrusion Detection, Third International Workshop, RAID 2000*, pages 145-161, Toulouse, France, October 2-4 2000. Lecture Notes in Computer Science #1907, Springer-Verlag, Berlin.
14. Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, and Ronald A. Olsson. A Methodology for Testing Intrusion Detection Systems. *Software Engineering*, 22(10):719-729, 1996.
15. Salvatore Stolfo, Wei Fan, Andreas Prodromidis, Wenke Lee, Shelly Tselepis, and Philip K. Chan. Agent-based Fraud and Intrusion Detection in Financial Systems. Technical report, 1998. Available at:
http://www.cs.columbia.edu/ wfan/research.html.
16. John W. Tukey. *Exploratory Data Analysis*. Addison Wesley College, 1997.

User Interaction Design for Secure Systems

Ka-Ping Yee

University of California, Berkeley
ping@zesty.ca

Abstract. The security of any system that is configured or operated by human beings depends on the information conveyed by the user interface, the decisions of the users, and the interpretation of their actions. This paper establishes some starting points for reasoning about security from a user-centred perspective: it proposes to model systems in terms of actors and actions, and introduces the concept of the subjective *actor-ability state*. Ten principles for secure interaction design are identified; examples of real-world problems illustrate and justify the principles.

1 Introduction

Security problems are often attributed to software errors such as buffer overruns, race conditions, or weak cryptosystems. This has focused much attention on the correctness of software implementations. However, correct use of software is just as important as correctness of the software itself. For example, there is nothing inherently incorrect about a program that deletes files. It is only when such a program deletes files against our wishes that we perceive a security violation.

It follows that the security properties of any system can only be meaningfully discussed in the context of the system's expected behaviour. Garfinkel and Spafford give the definition: "A computer is secure if you can depend on it and its software to behave as you expect" [7]. Notice that this definition is necessarily dependent on the meaning of "you", which usually refers to the user. It is impossible to even describe security without addressing the user perspective.

Among the most spectacular of recent security problems are the e-mail attachment viruses. Many of these are good real-life examples of security violations in the absence of software errors: at no point during their propagation does any software behave differently than its programmers would expect. The e-mail client correctly decodes the attachment and the system correctly executes the virus when the user opens the attachment. Rather, the problem exists because the functionally correct behaviour is inconsistent with what the user would want.

This paper aims to make two main contributions: first, it presents a model to guide thinking about this type of issue; and second, it gives specific recommendations in the form of ten interaction design principles for secure systems.

Among many designers, there is the pervasive assumption that improving security necessarily degrades usability, and vice versa; the decision to favour one or the other is typically seen as a regrettable compromise. For example, commonly suggested security fixes involve having the computer ask for user confirmation, yet we are also warned against annoying the user by asking too frequently [14]. In the end, these judgement calls are often made arbitrarily. A coherent interaction model will help designers resolve these dilemmas.

I take the apparently radical position that security and usability are not fundamentally at odds with each other. In fact, it should be clear upon reflection that the opposite makes more sense: a more secure system is more controllable, more reliable, and hence more usable; a more usable system reduces confusion and is thus more likely to be secure. Security advocates and usability advocates both want computers to correctly do what users want – no more and no less[1].

The results presented here come from discussing design challenges and user experiences at length with designers and users of software intended to be secure. After much debate and several iterations of refinement, we have tried to form a concise set of design principles that covers many of the important failure modes.

This paper is heavily abridged due to last-minute space restrictions; for a complete version with detailed case studies, **please see http://zesty.ca/sid/**.

2 Related Work

There seem to be relatively few development efforts in computer security [10] [12] [25] that have seriously emphasized user interaction issues. The Adage project [25], a user-centred authorization service, is probably the largest such effort to date. There have been several important usability studies of security applications [1] [13] [16] [24], all of which have shown the devastating impact that ignoring usability issues can have on the effectiveness of security measures. To my knowledge, this paper is the first attempt to propose a structured framework for design thinking and to suggest widely applicable guidelines for secure interaction design as opposed to studying a single application or mechanism.

Simultaneously addressing all of the principles presented here is admittedly a significant design challenge. Lest they seem too idealistic to be satisfiable, it is worth mentioning that there is an independently developed prototype of a secure desktop shell [3] that largely succeeds in satisfying most of these principles.

3 Design Principles

The following sections present a preliminary set of guidelines for secure interaction design. They are a snapshot of an ongoing process of refinement; applying them in practice will help to assess their completeness. Sufficiency cannot be proved, as it is impossible to guarantee the success of a user interface. So, our criterion for admitting something as a basic principle is that it be a *necessary*

[1] Often a dilemma stems from conflicts between what different *people* want. For example, some digital rights management efforts would make media content harder to use – but the resulting conflict is not one of security versus usability. It is actually a conflict between the desires of the users and the content providers. Balancing such conflicts is indeed an important problem, but outside of the scope of this paper. We will not address the design of systems that serve two masters, but understanding how to serve one master faithfully remains an important and necessary first step.

and *non-trivial* concern. I will argue that each principle is necessary by showing how its violation would likely yield a security vulnerability, and show that the principles are non-trivial by pointing out that violations exist in real software.

The *principle of least privilege* [21] is a starting point for our reasoning. It is better to think of our design principles in the context of least-privilege systems, rather than current systems not designed in a least-privilege style. A language-based security system, such as Java's "sandbox", is one kind of model in which one could hope to satisfy these principles. Other platforms designed around the least-privilege concept include the E language [5], KeyKOS [8], and EROS [22].

The design principles are listed here, with detailed explanations to follow. In the statement of these principles, the term "actor" is used to mean approximately "user or program", but this term will be explained more precisely below. The term "authority" refers to the ability to take a particular action.

Path of Least Resistance. The most natural way to do any task should also be the most secure way.

Appropriate Boundaries. The interface should expose, and the system should enforce, distinctions between objects and between actions along boundaries that matter to the user.

Explicit Authorization. A user's authorities must only be provided to other actors as a result of an explicit user action that is understood to imply granting.

Visibility. The interface should allow the user to easily review any active actors and authority relationships that would affect security-relevant decisions.

Revocability. The interface should allow the user to easily revoke authorities that the user has granted, wherever revocation is possible.

Expected Ability. The interface must not give the user the impression that it is possible to do something that cannot actually be done.

Trusted Path. The interface must provide an unspoofable and faithful communication channel between the user and any entity trusted to manipulate authorities on the user's behalf.

Identifiability. The interface should enforce that distinct objects and distinct actions have unspoofably identifiable and distinguishable representations.

Expressiveness. The interface should provide enough expressive power (a) to describe a safe security policy without undue difficulty; and (b) to allow users to express security policies in terms that fit their goals.

Clarity. The effect of any security-relevant action must be clearly apparent to the user before the action is taken.

3.1 The User and the User Agent

Thus far, we have mentioned "the user" several times, so it is necessary to precisely define what is meant by the user. For the purpose of this discussion, the user is a person at a computer using some interface devices such as a keyboard, mouse, and display. We are concerned with the software system that is intended to serve and protect the interests of the user, which is called the *user agent*.

On a single-user system, the user agent is the operating system shell, through which the user interacts with an arena of entities such as files and programs. On a multi-user system, other users use their own user agents to interact in the same arena. When the system is connected to the Internet, there is a new level of interaction. Now, the arena of the single computer is nested within the larger arena of the Internet. A new kind of user agent (such as a Web browser) now represents the user's interests in that larger arena of interacting entities. But in the smaller arena of the single computer, a Web browser is merely one of the participants, and the user's interactions with it are mediated by the lower-level user agent, the system shell. The browser might be used to contact yet a third user agent, such as a Web-based interface to a bank, operating in yet a third arena (of financial transactions among account holders), and so on.

This distinction is explained mainly to avoid confusion among levels of user agents. We will not directly address communicating through multiple user agents here; we consider only one level at a time. The rest of this paper discusses the design of any user agent serving a user. The ten design principles can apply to all kinds of users – not just end users of applications, but also system administrators or programmers, using whatever software they use for their tasks. Different users will have different expectations and needs, so the design of any secure system must begin with a clear understanding of those needs.

Principle of the Path of Least Resistance. In the real world, there is often no relationship between how safe or unsafe actions are, and how easy or hard they are. (It takes more concentration to use a hammer safely than unsafely, for instance.) We all have to learn, by being told, by observing others, and often by making painful mistakes, which ways of doing things are safe. Sometimes, through the design of our tools, we can make it easier to do things safely. Most food processors have a switch that lets them run only when the lid is closed. On power drills, the key for opening the drill chuck is often attached to the power cord so that unplugging the drill is a natural prerequisite to changing the drill bit. In both cases, a bit of cleverness has turned a safety precaution into a natural part of the way work is done, rather than an easily forgotten extra step.

Most users do not spend all their time thinking about security; rather, they are concerned with accomplishing some useful task. It is human nature to be economical with the use of physical and mental effort, and to choose the "path of least resistance". This can cause the user to work against security measures, either unintentionally or intentionally. The primary consideration is to keep the user's motivations and the security goals aligned with each other.

There are three aspects to this. First, observe that the ultimate path of least resistance is for the user to do nothing. Therefore, the default settings for any software should be secure (this is Saltzer and Schroeder's principle of "fail-safe defaults" [21]). It is unreasonable to expect users to read the documentation to learn that they need to change many settings before they can run software safely.

Second, consider how a user might work against security measures unintentionally. User behaviour is largely guided by perceived *affordances* (the visual and non-visual properties of the interface that suggest the available modes of interaction) [6] [19]. For example, if a button for a security function does not look clickable, the user might never notice that it is an available action.

Third, consider whether users will subvert security intentionally. If operating securely takes too much effort, users may decide to circumvent or ignore security measures while completely aware that they are doing so. There is a security risk in systems where the secure patterns of usage are inconvenient: each inconvenience increases the probability that the user will decide to operate unsafely.

All of these aspects can be summarized by the principle of the path of least resistance: the natural way should be the secure way.

Sometimes security goals might seem to oppose the desire to make things easy. However, these are truly in conflict less often than one might think. Tightening security has to do with having more specific information about the user's intent so it can be achieved safely. Often this information is already conveyed in the user's actions; it just needs to be applied consistently to improve security[2].

In the few remaining situations where it is absolutely necessary to introduce a new inconvenience for the sake of security, provide a payoff by making productive use of the extra information the user is asked to provide. For example, consider a multi-user system that requires a login procedure. Entering a password is an extra step that is necessary for security, but has nothing to do with the user's intended task. However, the login information can be used to personalize the experience – by providing a custom desktop, menu of favourite programs, and so on – to offset the added inconvenience. This helps keep users from trying to circumvent the login process (or choosing to use a system that doesn't have one).

3.2 Objects, Actors, and Actions

To interact with the world around us, we build a mental model of how it works. The model enables us to predict the consequences of our actions, so we can make useful decisions. Most concepts in the model fall within the two fundamental categories of *objects* and *actions*. This division is reflected in the way that nearly all languages, natural or invented, draw the distinction between nouns and verbs.

Some objects are inert: their behaviour is simple enough to be completely predicted using physical laws. For instance, if a cup is pushed off of a table, we expect it to fall to the ground. In Dennett's terms, our model adopts the *physical stance* [4] toward the cup. On a computer, one might consider a text file such an object. One can perform actions on the text file (say, copy or delete it) with simple consequences, but the file does not appear to take actions of its own.

Some objects have their own behaviours; we will call such objects *actors*, since they are capable of taking action. Even though such objects exist in the physical world and still follow physical laws in principle, their behaviour is too complex to model using only physics. Since we cannot predict exactly what an actor will do, we proceed by estimating reasonable bounds on its behaviour.

To a computer user, an application program is an actor. There are some expectations about what the program will do, and some limits on what it should be able to do, but no user could know exactly what program instruction is being executed at a given moment. Even though the operation of the program may be completely deterministic, we cannot take a physical stance toward it because it is too complex. Instead, we base our model on an understanding of the purpose for which it was designed – Dennett calls this taking the *design stance.*

[2] See the file browser example in the section on explicit authorization.

Other users are also actors. However, rather than having been designed for a purpose, their behaviour is directed by their own motivations and goals. As they are conscious entities, we model their behaviours in terms of their beliefs and intentions; that is, we adopt what Dennett calls the *intentional stance.*

Incomplete knowledge of the design, beliefs, or intentions of an actor produces uncertainty. We limit this uncertainty by applying the physical stance. For example, while one is inside a locked house, one has no need to model the intentions of any people outside the house because one is relying on the physical properties of the house to keep them out of the model.

Building models of actors is something we humans are very good at. Bruce and Newman [2] have examined in detail how the comprehension of "Hansel and Gretel" requires us to model actors, actors' models, actors' models of other actors' models, and so on, many levels deep – yet such complex modelling is a routine skill for young children. There is also evidence from computer-human interaction research that people perceive computers as "social actors" [20] even though machines do not actually possess human motivations. Both these reasons suggest that we indeed form our mental models of computers in terms of actors and actions. It is no coincidence that this is reminiscent of OOP, since the designers of Smalltalk also sought to match our mental models of the world [11].

Given this foundation, we can now formulate a more precise interpretation of Garfinkel and Spafford's definition of computer security. Our new definition is: "A system is secure from a given user's perspective if the set of actions that each actor can do are bounded by what the user believes it can do."

3.3 The System Image and the User Model

When a designer creates a system, th designer does so with some model in mind. But the designer doesn't get to communicate directly with the user. Rather, the designer decides how the system will work, the system presents an image to the user, and the user builds a model from interacting with the system. Communication of the model occurs only via this *system image.*

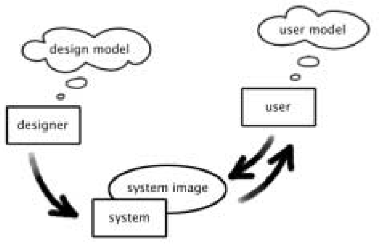


Fig. 1. The designer, the system, and the user (from [19]).

3.4 Aggregation

The actual working of a computer system is very complex and involves a great many small components and operations. To make the system comprehensible, the system image aggregates objects and actions into a smaller number of units.

Objects may be grouped by related concept or purpose. All the individual bytes of a file are usually taken together, given a single name, and presented as a single manipulable object. Actions may be grouped by concept, by locality in time, or by causality relationships. For example, while a request to open a Web page may involve many steps (looking up a hostname, opening a connection, sending a request, and so on) it is presented as a single action.

Most user interfaces allow the user to control some grouping in order to reduce their own mental effort. For instance, in most desktop operating systems, one can move a collection of files into a directory, and then move, copy, or delete the entire directory with a single operation: users can perform *subjective aggregation* [15] on the file objects. Systems with end-user programming features, such as macros, allow the subjective aggregation of several actions into a single action.

Principle of Appropriate Boundaries. Aggregation is important because it defines the terms in which authorities can be expressed. The user's model deals with concepts such as "actor X can perform action Y on object Z". The boundaries of the objects and actions are found by observing the system image, which conveys these boundaries through the methods it provides for identifying objects, communicating with actors, taking actions, and so on.

Here is an example to demonstrate the significance of choosing boundaries. Consider the idea that a secure operating system should let the user control the granting of authorities to programs. If a program spawns multiple processes, must the user separately grant authorities to each process? Or if a program relies on software modules or shared libraries, should the user have to separately control the authorities of every module? No: we resolve the dilemma by declaring that the boundaries between actors in the system image (which are also the boundaries of authority control) should be consistent with distinctions that the user cares about. Any boundary that could have meaningful security implications to the user should be visible, and those that do not should not be visible.

Stated another way, the interface should distinguish objects and actions along boundaries that matter to the user. If the distinctions are too detailed, there is greater risk that users will overlap or leave out specifications. However, if the boundaries are too few, users will be forced to give away more authority than they intend. The right distinctions can be discovered by asking: Would the user ever want to manipulate this authority independently of another? To grant authority to this actor but not another? To permit access to this resource but not another?

Supporting good distinctions sometimes places requirements on the software system behind the user interface. In the case of our example, since it would be infeasible to insist on separate control of authorities for each software component, the system should support the safe aggregation of components into useful conceptual units (that is, applications), such that reasoning about applications as individual actors holds valid. It follows that the system should enforce the corresponding boundaries: whenever two applications use the same software module, that module should be unable to convey authority between the applications.

3.5 The Actor-Ability State

Among other things, the user model contains some knowledge of actors and their abilities. As a starting point for talking about the user's conceptual state, let us consider a very simple model where the user knows about a finite set of actors $A = \{A_0, A_1, A_2, \ldots, A_n\}$ that can have an effect on the system, where A_0 is the user and there are n other actors. Each actor A_i is associated with an alleged set of potential actions, P_i. One can think of P_i as the user's answer to the question, "What can A_i do that would affect something I care about?" The knowledge of actors and abilities then consists of $\{\langle A_0, P_0\rangle, \langle A_1, P_1\rangle, \langle A_2, P_2\rangle, \ldots, \langle A_n, P_n\rangle\}$.

Since the user believes P_0 to be the set of available actions he or she can perform, the user will always choose to do actions from that set. In order for the user to choose actions that are actually possible, P_0 should be a subset of the user's real abilities. Since the user believes that P_i (for $i > 0$) is the set of available actions some other actor A_i can perform, the user expects that any action taken by A_i will be a member of P_i. To uphold this expectation, P_i must be a superset of that actor's real abilities. If we write R_i for actor A_i's set of real abilities, our *no-surprise condition* can be summarized as follows:

$$P_0 \subseteq R_0 \qquad \text{and} \qquad P_i \supseteq R_i \text{ for } i > 0$$

Principle of Explicit Authorization. It is essential to keep the actor-ability state in the user's model accurate at all times, since the user will make security-relevant decisions based on this state. To stay synchronized with reality, the user must be in control of changes in the actor-ability state. To maintain $P_i \supseteq R_i$, we require that only explicit user action can cause R_i to come to exceed P_i.

Explicit authorization is perhaps the most basic requirement for controlling authority in any system, and is a direct descendant of Saltzer's principle of least privilege. Requiring each authority to be explicitly granted increases the likelihood that actors will operate with the least authority necessary. Without such a restriction, the user becomes responsible for finding a potentially unlimited set of implicitly granted authorities to disable before the system is safe to use. In current systems, applications often have complete access to the network and filesystem without ever having been explicitly granted these authorities.

At first glance, it may seem that this principle is in conflict with the principle of the path of least resistance. Must we constantly intercept the user with annoying security prompts to confirm every action? No – most of the time, the user already provides plenty of information in the course of performing tasks. The system must merely honour the manipulations of authority that are already being communicated. For example, if the user asks an application to open a file and makes a selection in a file browser, it is already clear that they expect the application to read the file. No further confirmation is necessary. The single act of selecting the file should convey both the identity of the chosen file and the authority to read it. In many situations, combining designation with authority [9] yields an effective solution that improves both security and usability.

One can judge when explicit authorization is necessary on the basis of user expectations. For example, if there is a window that clearly belongs to an editor, one can expect the editor to draw in the window. However, it would certainly be unexpected for the editor to spontaneously delete the user's files. Just as it normally requires an explicit action for the user to delete files, so should it require explicit user action for another actor to acquire the ability to delete them.

The judgement of which authorizations should be explicit should be based on the potential consequences, not on the technical difficulty of the granting decision. *Any* authority that could result in unexpected behaviour should be controlled by the user. If the user cannot readily understand the consequences of granting an authority, then it should never be granted at all, not merely hidden under some "Advanced" section of the interface. If a truly necessary authority seems to require an unusual degree of technical knowledge, then the model presented to the user probably needs to be rethought in simpler terms.

Principle of Visibility. If the actor-ability state begins as a known quantity (some safe minimal set of authorities), and we are in control of each change in state, then in theory we know enough to ensure that our state remains accurate. However, there will be situations where one comes upon a new system in an unknown state. Moreover, it is unreasonable to expect a user to keep a perfect record of all grantings; human memory is fallible and limited in capacity. So we must enable users to update the actor-ability state in their heads at any time.

This is not to say that the interface should display all the low-level authorities of all the components in the system as a debugger would. Rather, it should show the right information for the user to ascertain the limits of what each actor can do, and should do so in terms of actors and actions that fit the user's model.

Visibility of system state is advocated as essential for usability in general [18]. Likewise, visibility of authorities is necessary for users to understand the security implications of their actions. It makes sense to show the actor-ability state in terms of the granting actions that brought it about. Past granting actions having no effect on the current state (such as access given to applications that have terminated) need not be visible. It is helpful to identify authorities by inspection of either the holder or the accessible resource. Without visibility of authorities, any application could retain and use a granted authority undetected and indefinitely, once the user has forgotten about the granting action.

Windows and Unix systems typically run dozens of background processes. It should be emphasized that this principle does not require the interface to display all these processes. Processes like the kernel swap daemon are not part of the typical user's model and so should not be considered actors. Consequently, system behaviour should maintain consistency with a model where such processes are not actors: the system must strive to maintain the appearance that the swap daemon has no effect on the user's world of files and programs.

One of the most widely publicized examples of a harmful background process is the "Back Orifice" program released by Cult of the Dead Cow in 1998. This program *is* an actor since it can modify files and transmit them over the network without user initiation, and therefore should be visible in the interface. Although Microsoft denied [17] that there was a Windows security issue here, the fact that Windows allows Back Orifice to run invisibly is what makes it so dangerous.

Principle of Revocability. To keep the actor-ability state manageable, the user must be able to prevent it from growing without limit. Therefore, wherever possible, the interface should allow the user to revoke granted authorities.

Another argument for facilitating revocation is the need to accommodate user error. It is inevitable that mistakes will be made; any well-designed system should help recover from them. In the context of granting authorities, error recovery amounts to revocation. One might intentionally grant an authority to an program and later discover that the program is untrustworthy; or one might inadvertently grant the wrong authority by mistake. In both cases, the granting decision should be reversible. Note that revocation prevents *further* abuse of an authority, but it is not always possible to undo damage caused by abuse of an authority while it was available. Thus, interfaces should avoid drawing an analogy between “revoke” and “undo”; “revoke” is better described as “desist”.

Principle of Expected Ability. Whereas the preceding three principles deal with managing other actors' abilities, the perception of the user's own abilities can also have security consequences. Users sometimes make decisions based on the expectation of future abilities. There can be serious security consequences if these expectations are wrong. The false expectation of an ability might give the user a false sense of security, or cause the user to make a commitment that cannot be fulfilled. Explicit authorization addresses one half of the no-surprise condition; this principle addresses the other half: $P_0 \subseteq R_0$.

For example, consider a system where granted authorities are usually revocable. If the user encounters an authority that cannot be revoked, the interface should make this clear, as it could affect the user's granting decisions.

3.6 Input and Output

Observation and control is conveyed through input and output, so the ability to use a system securely relies on the integrity of the input and output channels.

Principle of the Trusted Path. The most important input and output channels are those used to manipulate authorities; if these channels can be spoofed or corrupted, the system has a security vulnerability. Hence the principle of the trusted path: the user must have an unspoofable and incorruptible channel to any entity trusted to manipulate authorities on the user's behalf.

The authority-manipulating entity could be a number of different things, depending on the domain. In an operating system, the user needs a trusted path to the interface components for handling permissions and authentication. Microsoft Windows, for example, provides a trusted path to its login window by requiring the user to press Ctrl-Alt-Del. These keys cause a non-maskable interrupt that can only be intercepted by the system, thus guaranteeing that the login window cannot be spoofed by any application. In a language system for running untrusted code, such as Java, this issue also needs to be addressed.

Principle of Identifiability. The ability to identify objects and actions is the first step in proper communication of intent. When identity is threatened, either by inadvertent collision or by intentional masquerading, the user is vulnerable to error. Identification has two aspects: *continuity* (the same things should appear the same) and *discriminability* (different things should appear different).

That something is perceived to have an identity depends on it having some consistency over time. When we see an object that looks the same as something we saw recently, we are inclined to believe it is the same object. If an untrusted program can cause an object to look the same as something else, or it can change the appearance of an object in an unexpected way, it can produce confusion that has security consequences. The same is true for actions, in whatever way they are represented; actions are just as important to identify and distinguish as objects.

Note that it is not enough for the representations of distinct objects and actions to merely *be* different; they must be *perceived* by the user to be different. Even a choice of typeface can have security consequences. It is not enough for two distinct identifiers to be distinct strings; they must be displayed with visually distinct representations. In some fonts, the lowercase "L" and digit "1" are very difficult to distinguish. With Unicode, the issue is further complicated by characters that combine to form a single accented letter, which can cause different character sequences to be rendered identically on the screen.

It is not safe to assume that all actors will choose unique and consistent representations, so the continuity and discriminability of objects and actions are things that must be enforced by the system. This is the principle of identifiability.

Principle of Expressiveness. Sometimes a security policy may be specified explicitly, as in a panel of settings; other times it is implied by actions in the normal course of performing a task. In both cases, there is a language (of settings or actions) through which the user expresses a security policy to the system.

If the language used to express policies does not match the user's model of the system, then it is hard to set policy in a way that corresponds with intentions. In order for the security policy enforced by the system to be useful, we must be able to express a safe policy, and we must be able to express the policy we want.

For a good example of an expressiveness problem in real life, consider the Unix filesystem. Since each file can only be assigned to one group, it is impossible to share a file only for reading by one colleague and writing by another. Because the commands for setting permissions lack sufficient flexibility to express some kinds of sharing, users are sometimes forced to share files unsafely with everyone.

Principle of Clarity. When the user is given control to manipulate authorities, we must ensure that the results reflect the user's intent. We rely on correct software to enforce limits on each actor, but correctness of the implementation is irrelevant if the policy being enforced is not what the user intended. This can occur if the interface gives misleading, ambiguous, or incomplete information.

The interface must be clear not only about granting or revoking authorities; the consequences of any security-relevant decision, such as a decision to reveal sensitive information, should be clear. All the information needed to make a good decision should be accurate and available *before* the action is taken.

An interface can be misleading or ambiguous in non-verbal ways. Many graphical interfaces use common widgets and metaphors, conditioning users to expect certain unspoken conventions. For example, round radio buttons signify an exclusive selection of one option from many, while square checkboxes signify individual yes-or-no decisions. An ellipsis at the end of a menu command indicates that additional options must be specified before an action takes place, whereas the absence of an ellipsis implies that an action will occur immediately.

Visual interfaces often rely heavily on association between graphical elements, such as the placement of a label next to a checkbox, or the grouping of items in a list. Within a dialog box of security settings, we might rely on the user to correctly associate the text describing an authority with the button that controls it. Clarity can be evaluated in terms of the Gestalt principles of perceptual grouping [23], which suggest that visual associations are guided by proximity, closure, symmetry, figure-ground separation, continuation, and similarity.

3.7 Summary

In order to be able to use a system safely in a world of unreliable and adversarial software, a user must have confidence in all of the following statements:

- Things don't become unsafe all by themselves. (Explicit Authorization)
- I can know whether things are safe. (Visibility)
- I can make things safer. (Revocability)
- I don't choose to make things unsafe. (Path of Least Resistance)
- I know what I can and cannot do within the system. (Expected Ability)
- I can distinguish the things that matter to me. (Appropriate Boundaries)
- I can tell the system what I want. (Expressiveness)
- I know what I'm telling the system to do. (Clarity)
- The system protects me from being fooled. (Identifiability, Trusted Path)

4 Conclusion

I have argued that consideration of human factors is essential for security, and that security and usability do not have to be in conflict. In an attempt to provide some foundations for talking about secure interaction design, I have presented the actor-ability model and a set of design principles. The model is supported by evidence from other research; the principles are supported by direct reasoning, by the model, and by examples of problems in real software. The principles are applied in case studies of interaction problems, with proposed solutions, available at http://zesty.ca/sid/. I hope this paper will provoke discussion about a user-centred approach to computer security, and lead to computer systems that are safer and more reliable – not only in theory, but also in practice.

Acknowledgements

This paper builds directly on previous work with Miriam Walker. Many of the insights in this paper come from Norm Hardy, Mark S. Miller, Chip Morningstar, Kragen Sitaker, Marc Stiegler, and Dean Tribble, who participated in the extensive discussions during which the design principles were developed.

Thanks also to Morgan Ames, Verna Arts, Nikita Borisov, Jeff Dunmall, Tal Garfinkel, Marti Hearst, Nadia Heninger, Johann Hibschman, Josh Levenberg, Lisa Megna, David Wagner, and David Waters for help reviewing this paper.

References

1. A. Adams and M. A. Sasse. Users are Not the Enemy. In Communications of the ACM (Dec 1999), p. 40–46.

2. B. Bruce and D. Newman. Interacting Plans. In Readings in Distributed Artificial Intelligence. Morgan Kaufmann (1988), p. 248–267.
3. Combex. E and CapDesk: POLA for the Distributed Desktop. http://www.combex.com/tech/edesk.html.
4. D. Dennett. The Intentional Stance. MIT Press (1987).
5. ERights.org: Open Source Distributed Capabilities. http://www.erights.org/.
6. J. J. Gibson. The Ecological Approach to Visual Perception. Houghton Mifflin (1979), p. 127 (excerpt, http://www.alamut.com/notebooks/a/affordances.html).
7. S. Garfinkel and G. Spafford. Practical UNIX and Internet Security. O'Reilly (1996).
8. N. Hardy. The KeyKOS Architecture. In Operating Systems Review, 19(4)8–25.
9. N. Hardy. The Confused Deputy. In Operating Systems Review, 22(4)36–38.
10. U. Holmström. User-centered design of secure software. In Proceeedings of the 17th Symposium on Human Factors in Telecommunications (May 1999), Denmark.
11. D. Ingalls. Design Principles Behind Smalltalk. In BYTE Magazine (Aug 1981).
12. U. Jendricke and D. Gerd tom Markotten. Usability meets Security: The Identity-Manager as your Personal Security Assistant for the Internet. In Proceedings of the 16th Annual Computer Security Applications Conference (Dec 2000).
13. C.-M. Karat. Iterative Usability Testing of a Security Application. In Proceedings of the Human Factors Society 33rd Annual Meeting (1989).
14. K. Karvonen. Creating Trust. In Proceedings of the Fourth Nordic Workshop on Secure IT Systems (Nov 1999), p. 21–36.
15. M. S. Miller, C. Morningstar, and B. Frantz. Capability-Based Financial Instruments. In Proceedings of the 4th Conference on Financial Cryptography (2000).
16. W. S. Mosteller and J. Ballas. Usability Analysis of Messages from a Security System. In Proceedings of the Human Factors Society 33rd Annual Meeting (1989).
17. Microsoft. Bulletin MS98-010: Information on the "Back Orifice" Program. http://www.microsoft.com/technet/security/bulletin/ms98-010.asp (Aug 1998).
18. J. Nielsen. Enhancing the explanatory power of usability heuristics. In Proceedings of the ACM CHI Conference (1994), p. 152–158.
19. D. A. Norman. The Psychology of Everyday Things. New York: Basic Books (1988).
20. C. Nass, J. Steuer, and E. Tauber. Computers are Social Actors. In Proceedings of the ACM CHI Conference (1994), p. 72–78 (see http://cyborganic.com/People/jonathan/Academia/Papers/Web/casa-chi-94.html).
21. J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. In Proceedings of the IEEE, 63(9)1278–1308 (see http://web.mit.edu/Saltzer/www/publications/protection/).
22. J. Shapiro, J. Smith, and D. Farber. EROS: A Fast Capability System. In Proceedings of the 17th ACM Symposium on Op. Sys. Principles (Dec 1999).
23. M. Wertheimer. Untersuchungen zur Lehre von der Gestalt II. In Psychologische Forschung, 4, p. 301–350. Translation "Laws of organization in perceptual forms" in W. D. Ellis, A Sourcebook of Gestalt Psychology, Routledge & Kegan Paul (1938), p. 71–88 (see http://psychclassics.yorku.ca/Wertheimer/Forms/forms.htm).
24. A. Whitten and J. D. Tygar. Why Johnny can't encrypt. In Proceedings of the 8th USENIX Security Symposium (Aug 1999).
25. M. E. Zurko, R. Simon, and T. Sanfilippo. A User-Centered, Modular Authorization Service Built on an RBAC Foundation. In Proceedings of IEEE Symposium on Research in Security and Privacy (May 1999), p. 57–71.

Using Independent Auditors as Intrusion Detection Systems[*]

Jesus Molina and William Arbaugh

Department of Computer Science
University of Maryland
20742 College Park, MD
{chus,waa}@cs.umd.edu

Abstract. A basic method in computer security is to perform integrity checks on the file system to detect the installation of malicious programs, or the modification of sensitive files. Integrity tools to date rely on the operating system to function correctly, so once the operating system is compromised even a novice attacker can easily defeat these tools. A novel way to overcome this problem is the use of an independent auditor, which uses an out-of-band verification process that does not depend on the underlying operating system. In this paper we present a definition of independent auditors and a specific implementation of an independent auditor using an embedded system attached to the PCI bus.

1 Introduction

Computer systems have been made increasingly secure over the past decades. However, new attacks and the spread of harmful viruses have shown that better methods must be used. One approach gaining increasing popularity in the computer community is to use Intrusion Detection Systems (IDSs).

Intrusion Detection Systems identify attacks against a system or users performing illegitimate actions. Using a common analogy, having an Intrusion Detection System is like having a "burglar alarm" in your house. The alarm will not prevent the burglar from breaking into your house, but it will detect and warn you of the problem. Following the publication of the first research in Intrusion Detection Systems, a large number of diverse applications have been developed. One method of accomplishing this type of detection is the use of file system integrity tools. When a system is compromised, an attacker will often alter certain key files to provide continued access and to prevent detection. The changes could target any portion of the system software, e.g. the kernel, libraries, log files, or other sensitive files. File system integrity checkers detect those changes and trigger a corresponding alert. To guarantee the integrity of the file system, two approaches can be followed.

The first approach is to create a secure database, which is usually composed of hashes. The stored hash will be periodically checked against a newly computed hash. This method is used with tools such as Tripwire [1], Aide [2], and others.

[*] This work funded in part by DARPA grant #F306020120535

The second, more recent approach is to create digital signatures of sensitive data, such as executable files using asymmetric cryptography, and use these signatures to check the integrity of the signed file ([3], [4]).

Both approaches have advantages and drawbacks, but they share a common flaw: the auditing relies on the validity of the operating system. All the previous applications have made the assumption that the OS itself is not corrupted. Once the operating system is compromised the intruder can easily defeat integrity tools. As an example, in the Linux operating system, redirecting system calls using kernel modules can potentially compromise the system.

Also, since the binary of the Integrity Tool resides in the machine to be audited, the attacker may be able to corrupt the binary or the configuration files of the tool.

This work develops a novel way to overcome the problems of traditional Integrity Tools. Our approach is to use an independent auditor, i.e. a completely standalone and independent device, potentially tamper resistant, to perform the integrity detection checks.

2 Motivation

An Integrity Verification Tool that relies on the operating system of a penetrated machine can be easily deceived by corrupting the kernel. In fact, this problem is well known. In an article by ”Halflife” [5], a loadable kernel module was used to bypass the Tripwire integrity checking System. Since then, several tools for corrupting the operating system have been developed including Knark, famous for being used in the Ramen worm [6]. This section will explain the mechanics of these attacks. Although the attacks discussed here occurred on Unix-like operating systems, all operating systems are vulnerable to these kinds of attacks.

2.1 System Calls

User processes and the kernel run in different modes. The CPU itself enforces this policy. Every modern processor has at least two modes of operations, and in some cases, as in the x86, more than two. Every mode of operation allows some actions and does not permit others. In the case of the Unix operating system, only two levels are used: the lower, called user space or protected mode and the higher mode, called kernel space or supervisor mode. In this mode the process has unrestricted access to memory and devices. User-space applications are run in protected mode, while the kernel is executed in the supervisor mode. The only way an application will be able to access the sources restricted by the protected mode is through the kernel. If an application requests a service from the kernel, such as asking for more memory or accessing a hardware device, system calls are used to access the second mode of operation. These system calls, along with an interrupt reaching to the system, are the only ways to access kernel space.

In order to use system calls, the process will fill certain registers with appropriate values, including the type of system call to access, and then call a defined interrupt, dependent on the operating system and architecture. For example, in the Intel architecture the user process will call interrupt 0x80 if the operating system is Linux or interrupt 0x21 if the operating system is Windows. Then, depending on the system call used, the process will jump to a certain location of the kernel. The location in Linux is stored in a table (`sys_call_table`), where the addresses of the functions in the kernel are stored. The kernel will look at this table and jump to the corresponding address. After it returns from the call the kernel will do some system checks and continue in the address of the user space calling process.

2.2 Attacks in Kernel Space

In this section, we explore kernel attacks specific to the Linux operating system. Similar attacks could be launched in other Unix-like operating systems. The most straightforward way of changing the kernel is to replace the kernel binary itself. The kernel binary is usually placed in the /boot partition, so an attacker could compile his/her own version of the kernel and replace the binary. Some operating systems make this more difficult now, but the attack remains feasible on several current operating systems. Another possibility an attacker has is to use Loadable kernel Modules (LKMs). LKMs are a feature of Unix-like operating systems which allow dynamic changes to components of the kernel. An attacker will not have to recompile the complete kernel, but rather just code a LKM which can be loaded at any time and become part of the kernel.

Once the intruder has gained access to the kernel space, several attacks could be launched against the system so as to remain undetected. The most obvious attack is to redirect the system calls. Any program in user space such as Integrity Tools will use system calls to access kernel space, even for very simple operation such as reading a file. By redirecting the system call to a ”rogue” routine system call the attacker can hide the existence of any file in the system even from integrity checkers. Redirecting a system call using kernel modules is simple. As we have seen, the addresses where systems call will jump to when loaded are stored in a table. When the module is initialized, the kernel module will use code similar to the following:

```
ori_syscall=sys_call_table[SYS_sycall]
sys_call_table[SYS_syscall]=hacked_syscall
```

where hacked_syscall is a pointer to the function used to replace the system call. In the function hacked_syscall the attacker will call the original syscall and then change the results. For example, in Figure 1 the IDS never sees the file /foo/evilbinary because the system call filter eliminates it from the results.

```
res=(*ori_syscall)(parameters)
//change res to mislead the system
return(res)
```

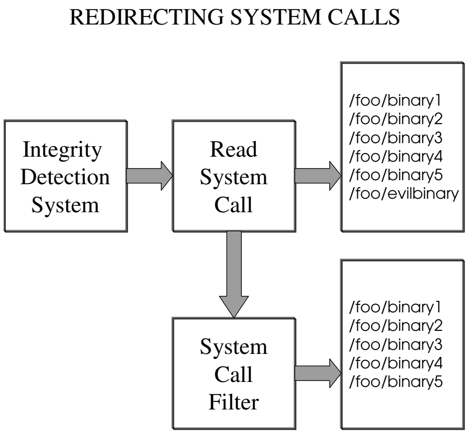


Fig. 1. Redirecting System Calls

Several Rootkits (a set of tools that an attacker uses to mask an intrusion and/or regain access later) take advantage of kernel modules. Other operating systems, such as Windows NT or 2000, may also be targets for these attacks, by using malicious system patches to the system or corrupted drivers. Some efforts have been made to counter the loading of kernel modules. Most of these techniques, as [7] in Windows NT, operate by restricting modules and drivers to be loaded or by using vigilant modules. However, the former creates a lack of flexibility that is usually not reasonable for most systems and the later tends to be a "chicken and egg" solution, as the attacker could modify his/her module to be loaded before the checking module, i.e. during the bootstrap.

3 Definition of Independent Auditors

In the present and following sections the terms "host processor" or "host system" will be used to define the machine or set of machines to be verified for file system integrity. The term "host" is slightly inaccurate. However, as in this work, the out-of-band verification system is implemented as an embedded coprocessor, and the term host processor will be used in order to avoid confusion. Other terms, such as host operating system, will be used throughout the text to refer to components of the system to be verified.

3.1 Properties

Machine A is an out-of-band auditor or independent auditor of Machine B if it accomplishes the following set of properties

1. *Unrestricted access*: Machine A must have unrestricted access to the internal devices of machine B to be verified or needed for the verification, including peripherals, hard disks and interrupts. Notice, however, that unrestricted write access, i.e. without mutual exclusion, to the internal components of the host system could lead to an unstable system.

2. *Secure transactions*: The channel used by the independent auditor to retrieve the data should be a secure channel, meaning a channel which cannot be eavesdropped or intercepted, nor modified.
3. *Inaccessibility*: Machine B must not have access in any way to the internal components of machine A, including memory and internal interrupts.
4. *Continuity*: Machine A must run immediately after machine B has setup the internal devices and is in a known trusted state. After this moment, Machine A must run continuously, independently of the behavior of machine B. Notice that power failures or hardware reboots should be the only way to restart machine A and must be labeled as high risk level alerts.
5. *Transparency*: The access to the internal devices should be transparent to the host system. However, concurrent access to the devices will probably occur unless mutual exclusion is provided. In these situations, the consequences to the host system should be minimized.
6. *Verifiable software*: All the code running in machine A must be trusted and verifiable. This, at least, implies that all running software in machine A must have the source code available. This includes the firmware, operating system and user space programs in machine A.
7. *Non-volatile Memory*: Machine A must be capable of retaining a record of the alerts even in the event of a power failure or reboot. Hence, machine A should have some non-volatile storage to record sensitive data.
8. *Physically secure*: Machine A should be physically secure.

3.2 Modes of Operation

The independent auditor has three different possible states. The running state is the normal mode of operation and is dependent on the method used for the integrity verification. The second state is the alarm state which is reached if an alarm is triggered in the running state. A final mode, which can be accessed only at boot time, is the management mode. This mode is only accessible through a set of secure mechanisms, and allows the administrator to change parameters in the secure coprocessor. An independent auditor is hence not vulnerable to API level attacks as described by Bond & Anderson in [8], as there is no interface to the independent auditor from the protected host. The running mode of operation could follow different methods to ensure the integrity of the data in the host machine, but the implementation described in section 4 uses a database to perform the integrity checking. All of these modes of operation assume that the host operating system is in a trusted state when the first checking takes place.

3.3 Audit Logs

Information pertaining to the alarm should be stored in a non-volatile storage device in the event of an alert. Using an independent auditor for integrity also creates an opportunity to not only store the information of the attack but also information before the attack. This is useful, as the auditor is not checking the system in real time. The auditor could log processes, measurements or events.

The auditor stores these sensitive logs in a trusted state. Every check without an alarm will ensure that the system remains in this trusted state. Hence, if the system audit occurs without an alarm, the auditor will update the data. In the event of an alarm, the data before the system compromise took place will be preserved, allowing the supervisor to retrieve the logs before the attack took place, in a trusted state. The recorded file could be compared to the files in the compromised machine to discover if the attacker has tampered with the logs and possibly uncover information about the type of attack and identity of the intruder. A discussion of the importance of secure audit logs can be found in [9].

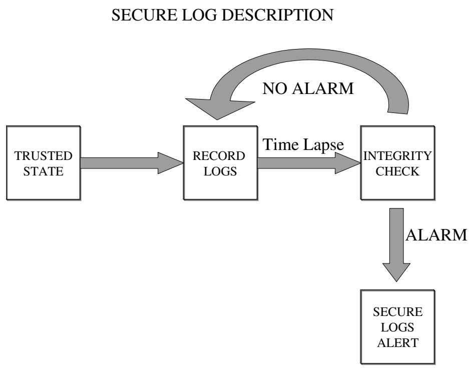


Fig. 2. Secure Logs Mechanism

3.4 Independent Auditor Using a Database Mode

This method does not differ much from its software counterpart, but as we see in Figure 3, all sensitive actions are moved inside the auditor and are therefore protected by the strong separation between the protected host and the auditor. The independent auditor will have a policy file, which is uploaded into the system in management mode, where the files to be checked will be declared along with the parameters to be verified. The files will be accessed periodically, and the set of actions stated by the policy file will occur. The independent auditor will retrieve the file's information, possibly computing its hash function. This information will then be checked against the locally stored information inside the auditor. If the information matches, the file has not been compromised. If it does not match, the alarm state will be triggered.

Using an independent auditor in database mode has several advantages as compared to its counterpart which is managed by the host operating system.

– The auditor handles the computational work. Hence, the performance of the host system is not degraded.
– The system is not vulnerable to subversion errors. Following directly from the inaccessibility property, an attacker gaining access to the host machine will not be able to corrupt the auditor.

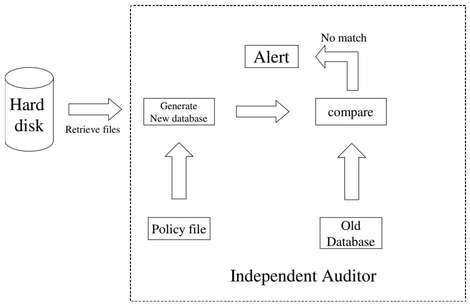


Fig. 3. Database Mode using an independent auditor

– Secure logs can be stored, which could greatly help in the forensics of the attack.
– In a system with more than one administrator a central administrator could make sure that the local administrators have not changed sensitive files.

4 Implementing an Independent Auditor

As an example of implementing an independent auditor, we used an embedded co-processor plugged directly into the PCI bus. The co-processor used was the SA-100 ARM processor with the 21285 core logic [10]. This coprocessor comes packaged as a EBSA-285 [11]. The EBSA-285 includes support for both volatile and non-volatile storage. The non-volatile storage is 4MB of flash ROM memory, and the volatile memory is upgradeable up to 256 MB, but for our system only 16 MB were necessary. The EBSA-285 is shaped as PCI card, and runs a romized Linux kernel. The Linux OS was chosen for the system as it supports a wide range of file-systems and it had good support for the SA-110 processor. AIDE was used as the application for performing the Integrity checks.

Our objective was to create a reliable integrity tool, which could be portable to any OS. We met the first requirement. We were able to mount the partition of a Linux box and a Windows NT box, and check the integrity of the file-systems using AIDE. However, since the access to the IDE controller through the PCI bus is non-atomic, when both the host OS and the EBSA-285 write to the registers on the IDE controller at the same time, a race condition occurs. To avoid these problems, patches for the host OS must be supplied to create a mutual exclusion mechanism.

4.1 The EBSA-285 as an Independent Auditor

In this section the suitability of the EBSA-285 to act as an independent auditor will be discussed. An assumption of this work is that the host machine is physically secure. The term physically secure is used in this section to describe a machine whose internals can not be tampered with by an attacker. The attacker, however, could have access to the peripherals attached to the host, including keyboard and monitor.

The EBSA-285 must have open access to the hard disk data to meet the property of unrestricted access. The EBSA-285 has access to the entire PCI bus, hence is capable of reading the registers and data from all PCI devices plugged into the same bus. If the IDE controller is plugged into the same bus as the EBSA-285, the EBSA-285 will be able to read the hard disk without the intervention of the host OS. Notice that if the EBSA-285 is plugged into a slot using a different bus than the IDE controller, the access will not be possible and the EBSA-285 will not be capable of acting as an independent auditor of the host processor. The EBSA-285 is not able to "listen" to the interrupts raised by the different devices in the PCI bus. These interrupts, however are not imperative to the auditing, as a polled method can be used to read and write data to the peripherals.

The channel used to retrieve the information from the peripherals is the PCI bus. This channel is secure as it is an internal part of the computer and we have assumed that the internal parts of the host are secure. Hence, the EBSA-285 accomplishes the property of secure transactions. The EBSA-285 does not map its memory (either ROM or RAM) via the PCI bus, which allows the host processor to access only mailbox registers and doorbell registers. The EBSA-285 uses these register as information, and does not interfere with its operation, hence not breaking the inaccessibility property.

Once the EBSA-285 starts auditing only a power failure or reset of the host machine will stop it from functioning, and these events will be labeled as alarms. The EBSA-285 will begin functioning after the host machine has set up all the internal peripherals. In our case, the EBSA-285 will begin functioning before this happens, so the EBSA-285 has a mechanism to stall its booting until all the peripherals have been configured. Therefore the property of continuity is satisfied. Because the EBSA-285 has direct access to the registers of the PCI devices, it is able to access the data by polling without the supervision of the host OS, and therefore satisfying the requirement of the property of transparency. Notice, however, that some mechanism should be implemented to avoid concurrent writes to the IDE registers, which would lead to an unstable system.

The software running in the EBSA-285 is open source. It is composed of a minimum bootloader and the ARM-port of the Linux operating system, with some changes to support the EBSA-285 and the polling method.

The EBSA-285 can use the flash ROM to store the alarms and logs. The ROM normally cannot be reprogrammed if the program is executed from the flash ROM. To avoid this problem, the bootloader copies the root file-system and the operating system to the RAM memory before executing it, freeing the flash ROM.

4.2 Solving Race Conditions

The EBSA-285 was able to access the IDE hard disk using a polled IDE driver instead of the usual Linux driver and was able to mount the hard disk independently of the host operating system used, as long as it was supported by the Linux operating system[12]. However, if the host operating system tried to access the hard disk at the same time as the EBSA-285, race conditions occurred, as requests to the IDE controller through the PCI bus are not atomic. The PCI specification [13] contains a method to achieve atomic transactions using the PCI bus with the LOCK# signal. This signal, however, is not used in common drivers and few motherboards support its use. While we hoped to avoid changes to the host operating system, we could not due to the lack of an atomic lock on the PCI bus.

To avoid the race conditions we used a communication path between the host processor and the independent auditor, named mailbox registers and doorbell interrupts. Using these tools will not break the inaccessibility property as the independent auditor will only use this signal as information to prevent the race conditions. The host processor is only able to modify these specific registers.

If the host accesses the hard disk, it will write a doorbell register in the EBSA-285, which will raise a sleep interrupt in the EBSA-285. If the EBSA-285 is engaged in any hard disk transaction, it will finish the current block transaction, and then send an ACK to the host using a mailbox register. After receiving the ACK the host will resume normal operation. Once the host has finished using the hard disk, it will raise a wakeup doorbell interrupt in the EBSA-285. The EBSA-285 will be then free to begin a new transaction. The transaction size depends of the mode of operation.

The behavior of both the EBSA-285 and the host processor can be summarized in the state machines shown in figures 4 and 5.

It could be argued that an attacker could easily halt the EBSA-285 by changing the driver so that the host will never send a wakeup interrupt to the EBSA-285. To combat this attack we use a counter, which will be set to the maximum transaction value and will wakeup the EBSA-285 after that time, even if the interrupt never arrives. Additionally, an alarm will be raised.

Another possible attack would be to prevent the host from sending the sleep interrupt to the EBSA-285. In this case, the EBSA-285 would proceed to read even if the host is accessing the hard disk. This could cause a corruption of the file-system. However, this attack is not only a problem with the EBSA-285, for if an intruder is able to change the drivers, obviously he/she would be able to corrupt the file-system anyway.

4.3 Implementation Results

To collect the timing information, the host used was an Intel Pentium III Coppermine. As stated before, the EBSA-285 uses a polled driver. The average throughput of the EBSA-285 to the hard disk using this polled driver is 1.40 Mb/s.

The performance varies greatly from one system to another, or even in the same machine: at different times the data to be checked could be stored in cache or not, the CPU could be burdened by a huge amount of processes or just a few, and so on.

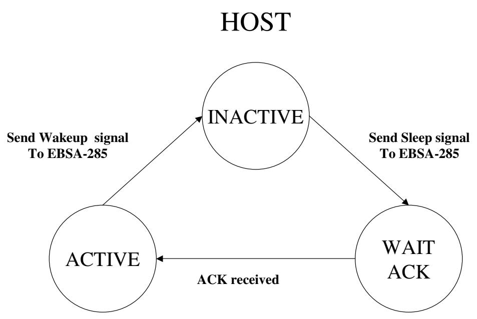


Fig. 4. Host Hard disk Access State Machine

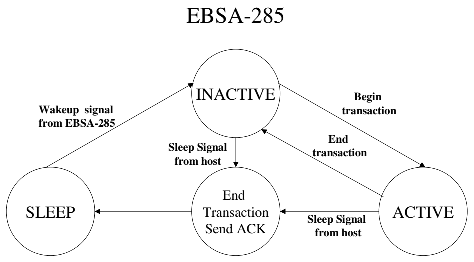


Fig. 5. EBSA-285 Hard disk Access State Machine

In the EBSA-285, the numbers are deterministic. The EBSA-285 always bypasses the internal cache and retrieves the data directly from the hard disk at a constant rate. The CPU is always running a minimal number of process, so the CPU is 99% devoted to computing the hashes and perform the different tests. In Table 1 we can see the comparison between the two unique states we can have, the EBSA-285 executing AIDE alone and the EBSA-285 executing AIDE at the same time as the host machine is performing a hard disk access, for three different amount of files. This hard disk access is performed using a "worst scenario" approach: we read a file from the host machine bigger than the RAM memory while AIDE is running on the independent auditor, therefore the data will never be cached.

The time spent in kernel space is roughly the time the EBSA-285 spent retrieving the data. This time is directly proportional to the total size of the data to be retrieved, while the amount spent in user space is proportional to the complexity of the hash function.

The overhead of the locking mechanism to the host machine when there isn't a concurrent access with the EBSA-285 is negligible, as the only addition to the driver is a new register write through the PCI bus (writing to a Mailbox Register in the EBSA-285).

The locking mechanism was created to give priority to the host machine. When the host machine begins a hard disk access, if the EBSA-285 is accessing the hard disk at the same time, the EBSA-285 will stall until the host machine has ended the request (the size of a request depends of the mode used to access

Table 1. Different Timings of executing AIDE on the EBSA-285 . The first column states if the host machine is accessing the hard disk concurrently with the EBSA-285. the second column the number of files and the total size of all of them. The third and fourth columns the amount of time spent in User Space and in kernel Space, respectively. The tests performed by AIDE were: permissions, inode, user, group, size and checksum using SHA1 checksum

Concurrent	File Amount and Size	User Space	System Space	Total
No	36 (7644k)	1.18s	3.69s	4.86s
Yes	36 (7644k)	1.26s	4.02s	5.03s
No	2302 (47552k)	7.12s	36.55s	43.68s
Yes	2302 (47552k)	7.06s	55.00s	62.71s
No	3484 (128736k)	17.53	75.51	93.29
Yes	3484 (128736k)	17.67	101.37	119.25

the hard disk). As a result the impact in the performance of the host machine is very low. In fact, most of the time the host machine does not access the hard disk, but reads the data stored in the internal cache. Even in the worst case scenario, where the hard disk is reading a file bigger that its memory while the EBSA-285 is performing several hard disk access, only supposes a maximum of a 5% overhead in the timing results of the host machine.

5 Conclusions

Current computer systems can not fully protect themselves against motivated attackers because of the attackers ability to change the underlying operating system once gaining system privilege. The attacker can simply change the operating system to “lie” to any security system– not only integrity detection systems.

In this paper, we proposed the notion of an independent auditor whose role is to serve as an unimpeachable reviewer of the state of the protected host. We defined the properties required for such an auditor, and we implemented such a device for measuring the integrity of a protected host. The results we achieved were excellent in that at most only five percent of overhead was added to the protected host– in the worst case.

While such a system is not required in all computer systems within an organization, the capabilities provided by an independent auditor are tremendously important to server systems. Given the low cost of these boards, $200, and the vast increase in protection they provide– organizations serious about protecting their resources should consider such protection.

Acknowledgments

The authors wish to thanks (Alphabetically) to Adam Agnew, Michael Barr, Jeffrey Chung, Jim Fischer, Virgil Gligor, Russel King, Arunesh Mishra and Charles Silio for the helpful feedback and comments during the course of this work.

References

1. G. H. Kim and E. H. Spafford: The design and Implementation of TRIPWIRE: A File System integrity checker. Technical Report, TR-93-071.
2. R. Lehti, P. Virolainen: AIDE (Advanced Intrusion Detection Environment), `Web ref: http://www.cs.tut.fi/~rammer/aide.html`
3. S. M Beattie, Andrew P. Black, Cristing Cowan, Calton Pu, Lateef P. Yang: Cryptomark: Locking the Stable door ahead of the Trojan Horse. Technical review, Jul 2000.
4. L. Van Doorn, G. Ballintijn and W. Arbaugh: Signed Executables for Linux. Technical review, 2000.
5. "Halflife": Bypassing Integrity Checking Systems. Phrack, volume 7, Issue 51 , September 1997.
6. CERT(r): em Incident Note IN-2001-01, Widespread Compromises via "ramen" Toolkit. January 18, 2001
7. Pedestal software, Integrity Protection Driver (IPD), `http://pedestalsoftware.com/intact/ipd/`
8. Mike Bond, Ross Anderson: API-Level Attacks on Embedded Systems. In IEEE Computer Vol. 34, No. 10 pp. 67-75, October 2001.
9. Bruce Schneier and John Kelsey: Secure Audit Logs to Support Computer Forensics. ACM transactions on Information and System Security, Vol.2, No 2, May 1999, Pages 159-176.
10. Intel Corporation: Datasheet, 21285 Core Logicfor the SA-110 Microprocessor. September 1998.
11. Intel Corporation: Reference Manual, StrongARM EBSA-285 Evaluation Board. October 1998.
12. M. Tanuan: An Introduction to the Linux operating system Architecture Web Reference
`http://www.grad.math.uwaterloo.ca/~mctanuan/cs746g/LinuxCA.html`
13. PCI Special Interest Group: PCI Local Bus Specification, Revision 2.2. December 1998.
14. Intel Corporation: User Manual, 21555 Non-Transparent PCI-to-PCI Bridge. July 2001.
15. Jim Fischer: CiNIC - Calpoly Intelligent NIC. EE MS Thesis, June 2001.
16. Xilinx: Virtex-II Pro Platform FPGA Handbook. January 31 2002.

Cellular Automata Based Cryptosystem (*CAC*)

Subhayan Sen[1], Chandrama Shaw[1], Dipanwita Roy Chowdhuri[2], Niloy Ganguly[3], and P. Pal Chaudhuri[1]

[1] Department of Computer Science & Technology, Bengal Engineering College (D U) Howrah, India 711103
{subhayan@,cshaw@,ppc@ppc.}becs.ac.in

[2] Department of Computer Science & Technology, Indian Institute of Technology Kharagpur, India 721302
drc@cse.iitkgp.ernet.in

[3] Computer centre, IISWBM, Calcutta, India 700073
n_ganguly@hotmail.com

Abstract. This paper introduces a *Cellular Automata* (CA) based symmetric key cryptosystem for block cipher. The scheme named as CAC (Cellular Automata based Cryptosystem) employs a series of transforms – simple, moderately complex, and complex – all generated with different classes of CA. CAC provides a low cost, high speed cryptosystem with desired level of security. Cryptanalysis of the proposed scheme is reported along with similar analysis for two popular systems - DES and AES. Experimental results confirm that the security of the system is significantly better than DES and comparable to that of AES. The encryption/decryption throughput is higher than that of both AES and DES.

1 Introduction

This paper reports a high speed, low cost cryptosystem with desired level of security. Its hardwired version supports real time encryption/decryption. The scheme referred to as CAC (Cellular Automata based Cryptosystem) employs different classes of transforms generated with Cellular Automata (CA).

We currently live in an internetworked society where a large volume of different classes of data travel around the globe. This electronic data transmission should be secured enough against unwanted interceptor. In the above context we aim to achieve the following objectives for design of CAC: (i) High speed operation, specifically on line real time data encryption/decryption; (ii) low cost of implementation; and (iii) acceptable level of security.

In this paper, we concentrate on developing an innovative cryptosystem based on the theory of *Cellular Automata*(CA). A CA can be viewed as a parallel machine simulating a discrete dynamical system. Further, the inherent parallelism of CA cells with their simplicity and local interactions make it particularly suitable for designing a low-cost crypto-hardware. The above mentioned advantages have lead researchers to design various Cellular Automata based cryptosystems

[2–5]. In [2] Cellular Automata is used as random sequence generator. In [3], non-homogeneous Cellular Automata has been proposed for *public-key* cryptosystems. Gutowitz [4] uses Cellular Automata as discrete dynamical system to add complexity of the cryptosystem. But none of these schemes has been able to withstand the modern attacks developed out of the cryptanalysis techniques [1]. Cellular Automata based block cipher and stream cipher schemes are also presented in [5]. But the scheme is insecure because of its inability to change the key. The ability to change the key is essential for any cipher. Also the scheme, as pointed out in [6], generates a subgroup of *affine group* and not the *alternating group.* In [7] another CA based block cipher scheme was proposed. But this is also unable to come out from the affine group constraint and so fails to achieve the desired level of security. This paper removes this bottleneck while generating non-affine CA transform.

The CA based cryptosystem (CAC) along with the encryption and decryption algorithm is outlined in *Section 3* after introducing CA preliminaries in *Section 2*. Discussion on cryptanalysis of CAC are covered in *Section 4*. A comparative study with other symmetric key block-cipher like DES and AES has also been included in this section. Finally, a low cost pipelined architecture of CAC crypto-hardware is reported in *Section 5*.

2 Cellular Automata Preliminaries

2.1 Introduction to $GF(2)$ CA

A CA consists of a number of cells arranged in a regular manner, where the state transitions of each cell depends on the states of its neighbors. The next state of a particular cell, as shown in Figure 1, is assumed to depend only on itself and on its two neighbors (left and right) and this leads to 3-neighborhood dependency. The state $q \in \{0,1\}$ of the i^{th} cell at time $(t+1)$ is denoted as $q_i^{t+1} = f(q_{i-1}^t, q_i^t, q_{i+1}^t)$, where q_i^t denotes the state of the i^{th} cell at time t and f is the next state function called the rule of the automata [8]. Since f is a function of 3 variables, there are 2^{2^3} or 256 possible next state functions. The decimal equivalent of the output column in the truth table of the function, as noted below is denoted as the rule number [8].

Neighborhood :	111	110	101	100	011	010	001	000	*RuleNo*
(i) NextState :	0	1	0	1	1	0	1	0	90
(ii) NextState :	1	0	0	1	0	1	1	0	150

A CA employing both XOR and XNOR local rules for different cells are referred to as *Additive CA*, while the ones using only XOR rules are noted as *Linear CA*. This class of CA is referred to as $GF(2)$ CA in the sense that each of the CA cells can store an element 0 or 1 in $GF(2)$. comprehensive treatment of $GF(2)$ CA results is noted in the book [8].

We next generalize this structure to study of $GF(2^p)$ CA [10] where each cell is capable of processing a symbol of $\{0,1,\cdots 2^p - 1\} \in$ GF(2^p). CAC employs $GF(2^p)$ CA that can be analyzed with the theory of extension field [9].

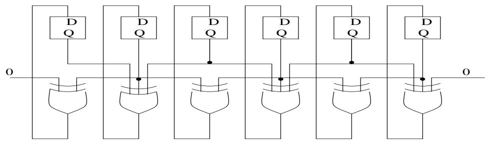


Fig. 1. A One Dimensional null boundary 6 cell Additive CA with rule vector <90,150,90,150,90,150>

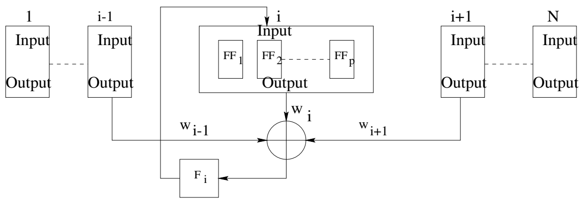


Fig. 2. General structure of a $GF(2^p)$ *Cellular Automata* machine

2.2 Introduction of $GF(2^p)$ CA

The *Fig.2* depicts the general structure of an n-cell $GF(2^p)$ CA. The connection among the cells follow a three neighborhood dependency in the sense that the next state $q_i(t+1)$ of the i^{th} cell depends upon the present states of $(i-1)^{th}$, i^{th} and $(i+1)^{th}$. The connection among the cells of the CA are weighted in the sense that to arrive at the next state $q_i(t+1)$ of i^{th} cell, the present states of $(i-1)^{th}$, i^{th} and $(i+1)^{th}$ are multiplied respectively with w_{i-1}, w_i and w_{i+1} and then added. In $GF(2^p)$ CA each cell, having p number of FFs(Flip-Flops), can store values 0,1,2,$\cdots$,(2^p-1) and the weights being elements of $GF(2^p)$.

If all the states in the state transition diagram of a CA lie in some cycles, it is a group CA; otherwise it is a non-group CA. Group CA can further be classified into *maximum* and *non-maximum* length CA. An n-cell *maximum*-length $GF(2^p)$ CA is characterized by the presence of a cycle of length (2^{pn}-1) with all nonzero states. On the other hand, a non-maximum length CA state transition diagram has a number of cycles.

3 Cellular Automata Based Cryptosystem (CAC)

The objectives of high speed of operation with lower implementation cost achieving high level of security are conflicting in nature. In order to meet such conflicting demands we apply a series of transforms of increasing complexity in successive levels. For the current vesion of CAC, four levels of transforms, as shown in *Fig. 3*, have been employed. The basic guiding factor is to achive a trade off in typical engineering design – realize the targeted objective with higher efficiency while minimizing cost. With a similar analogy, we apply low cost high speed linear and affine transforms in first two levels, while introducing complex non-affine transform at the third level to achive higher level of security. The fourth level is responsible for key-mixing.

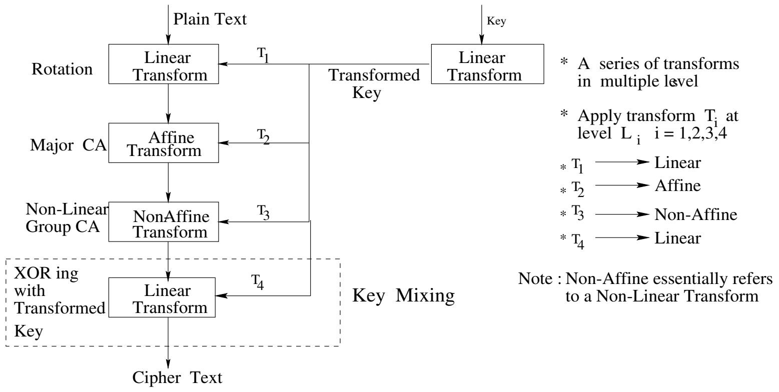


Fig. 3. Design of the encryption scheme

3.1 A Specific Implementation

A specific *CAC* implementation is shown in *Figure 4*. Four different levels of transformations are explicitly marked as *Level 1, 2, 3 and 4*. Different stages of computation are marked as (I), (II), (III), (IV), (V) and (VI) in *Figure 4*.

The encryption algorithm is based on two different classes of group *CA*, 16 cell $GF(2^8)$ *Major CA* and 16 cell $GF(2^8)$ *Minor CA*. The *Major CA* is a non-maximum length group *CA* with equal cycles of length 32. The *Minor CA* is a maximum-length *CA*. *CAC* with 16 cell $GF(2^8)$ *CA* can encrypt 16×8=128 bits of token at a time. Thus the token ($\mathcal{T}$) size and also the key size are taken as 128 bits.

Note: *The size of the key and token can be adjusted by changing the number of cells of the CA and/or the value of p in* $GF(2^p)$. *The basic scheme does not get effected by that.*

The operation of the encryption and decryption scheme is presented below. Each step of the operation is explained with the help of the *Figure 4*.

Level 1 – Linear Transformation on Key: The key (K) used for *CAC* scheme is a bit string of length same as the number of bits used for *Minor CA*. The input key is used as the initial seed of the *Minor CA*.

Role of *Minor CA***:** The *Minor CA* is operated for a fixed number of clock cycles (d) for input of each token. Initially, the seed of the *Minor CA* (S_0) is the key K (marked as I in the *Fig 4*). For each successive input token, *Minor CA* generates a new state (marked as S_N) after running d number of steps from its current state (shown as II in *Figure 4*). The state S_N is utilized for four different purposes:

1. Provides the value δ by which each byte of the input token ($\mathcal{T}$) is rotated.
2. Provides seed for equivalent *Major CA* synthesis.
3. Provides the number of clock cycles (Δ) of *Major CA* operation for encryption.

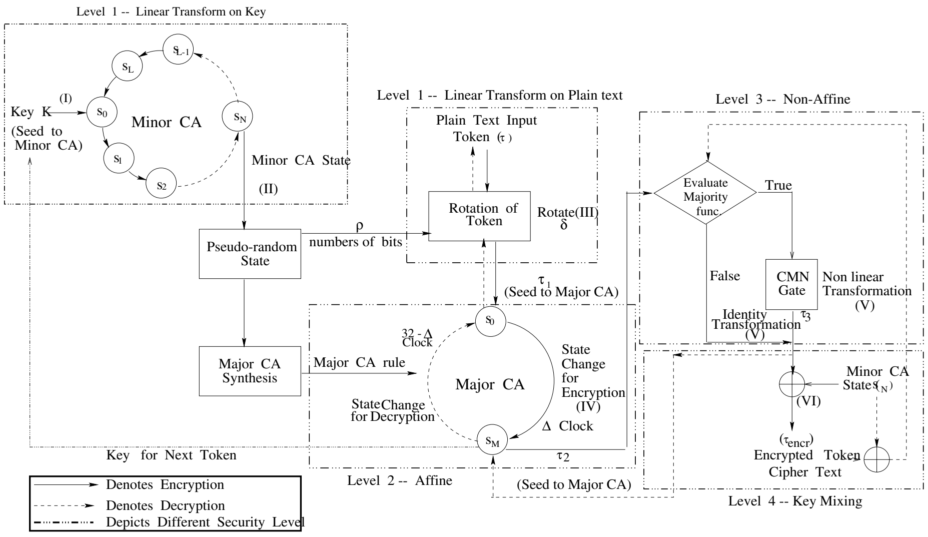


Fig. 4. Total encryption and decryption scheme

4. *XOR*ing the intermediate encrypted token to form the final encrypted token $\mathcal{T}_{encr}$.

Level 1 – Linear Transform on Token: The linear transformation of the token $\mathcal{T}$ to $\mathcal{T}_1$ is executed by rotating each byte of $\mathcal{T}$ by δ amount of steps (*III* in *Figure 4*). In decryption side token generated from *Major CA* is subjected to a same amount of rotation in the opposite direction.

Level 2 – Affine Transform: Next we give an affine transform to the token $\mathcal{T}_1$ by using the *Major CA*. The *Major CA* is generated at runtime by an efficient synthesis algorithm [10]. The *Major CA* uses the input token ($\mathcal{T}_1$) as its seed and operates for Δ number of cycles to generate the encrypted token $\mathcal{T}_2$ (*IV* in *Figure 4*). The *Major CA* has cycles of equal length 32. So, the *Major CA* will invariably return back the input token $\mathcal{T}_1$ after running for 32 number of clock cycles. So the original token is returned after running the *Major CA* for (32-Δ) clock cycles at the decryption side.

Level 3 – Non-Affine Transform: A non-affine transform is achieved by selective use of *Control Majority Not* (*CMN*) gate. *CMN* gate is a non-linear reversible gate with four inputs(1 data input and 3 control inputs) and one output. We will denote the control bits by c_1, c_2 and c_3. The function is defined as

$$y = x \oplus \{(c_1 \cdot c_2) \oplus (c_2 \cdot c_3) \oplus (c_3 \cdot c_1)\}$$

where $\oplus$ denote the *XOR* and $\cdot$ denote the *AND* function. The token $\mathcal{T}_2$ is subjected to *CMN* gate depending on the result of a function called *Majority Evaluation Function.* The Majority Evaluation Function takes the 5 bits, referred to as fixed-bits, of $\mathcal{T}_2$ and calculate the number of 1's in these bits. The 5 bit positions are selected depending on S_N(*Figure 4*). If the number of 1's is greater than 2 then each bit of $\mathcal{T}_2$ except these fixed-bits are subjected to CMN gate. Otherwise, $\mathcal{T}_2$ remains as it is. In any case, we call the resultant token as $\mathcal{T}_3$ (V in *Figure 4*). Two sets of control bits taken from S_N applied to the CMN gate alternately. The fixed-bits have to be remained fixed because during decryption the same fixed-bits will require to get the same result from majority evaluation function.

Level 4 – Key Mixing: To enhance the security and randomness, we generate final encrypted token $\mathcal{T}_{encr}$ by XORing the $Minor\ CA$ state S_N with the token $\mathcal{T}_3$(VI in *Figure 4*).

The algorithm for encryption and decryption process is presented next.

Algorithm 1 Encryption

```
Input: input file to be encrypted
K=key
Output: encrypted file
begin
    Step 1. Divide the file into 128 bit tokens ($\mathcal{T}$).
    Step 2. Load initial seed of Minor CA $S_0$=K
    For each token $\mathcal{T}$ begin loop
        Step 3. Run the Minor CA for d time steps and obtain $S_N$
        Step 4. Obtain $\delta$ from $S_N$. Rotate $\mathcal{T}$ by $\delta$ number of times and obtain $\mathcal{T}_1$
        Step 5a. Randomly synthesize Major CA ($CA_{maj}$) using $S_N$ as seed
        Step 5b. Obtain $\Delta$ from $S_N$
        Step 5c. Run $CA_{maj}$ for $\Delta$ time steps with $\mathcal{T}_1$ as seed to obtain $\mathcal{T}_2$
        Step 5d. $S_0$=$\mathcal{T}_2$
        if $\mathcal{T}_2$ satisfies MEF
            Step 6a. Obtain the 2 sets of Control bits for CMN gate
            Step 6b. Apply CMN gate to non-fixed bits of $\mathcal{T}_2$ using the Control bits
                     alternately
            Step 6c. Assign the result to $\mathcal{T}_3$
        end if
        Step 7. XOR $\mathcal{T}_3$ with $S_N$ to get $\mathcal{T}_{encr}$
        Step 8. write $\mathcal{T}_{encr}$ in output file
    Go to Step 3 untill the input file is exhausted
end
```

Algorithm 2 Decryption

```
Input: input file to be decrypted
K=key
Output: decrypted file
begin
    Step 1. Divide the file into 128 bit tokens ($\mathcal{T}_{encr}$)
    Step 2. Load initial seed of Minor CA $S_0$=K
    For each token $\mathcal{T}_{encr}$ begin loop
        Step 3. Run the Minor CA for d time steps and obtain $S_N$
        Step 4. XOR $\mathcal{T}_{encr}$ with $S_N$ to get $\mathcal{T}_3$
        if $\mathcal{T}_3$ satisfies MEF
```

```
                Step 5a. Obtain the 2 sets of Control bits for $CMN$ gate
                Step 5b. Apply $CMN$ gate to non-fixed bits of $\mathcal{T}_3$ using the Control bits
                         alternately
                Step 5c. Assign the result to $\mathcal{T}_2$
            end if
            Step 6a. $S_0=\mathcal{T}_2$
            Step 6b. Randomly synthesize Major CA ($CA_{maj}$) using $S_N$ as seed
            Step 6c. Obtain $\Delta$ from $S_N$
            Step 6d. Run $CA_{maj}$ for 32-$\Delta$ time steps with $\mathcal{T}_2$ as seed to obtain $\mathcal{T}_1$
            Step 7. Obtain $\delta$ from $S_N$. Rotate $\mathcal{T}_1$ by $\delta$ number of times in reverse order
                    and obtain $\mathcal{T}$.
            Step 8. Write back $\mathcal{T}$ into output file
        Go to Step 3 untill the input file is exhausted
    end
```

4 Analysis of *CAC*

4.1 Different Levels of Security

Large Key Space: The number of possible key is very large (2^{128}) and all key are equiprobable to occur. This randomness in key generation gives random probability distribution in key space. Since we can change the size of the minor and major CA the key size can also vary. So we can have a variable key space of any arbitrary size.

Security Level 1 – Linear Transformation: Each byte of token $\mathcal{T}$ is subjected to a random rotation decided by $Minor\ CA$ state. Since $Minor\ CA$ is an excellent pseudo-random generator [8], this rotation of token introduces a degree of randomness to the input token.

Security Level 2 – Affine Transformation and On-line Synthesis of Major CA: The state transition of a $Major\ CA$ which is additive generates an *affine transformation.*

On the fly generation of $Major\ CA$ reduces the memory requirement by a large amount and as well as enhances the security. The number of all possible CA having the cycle structure of $Major\ CA$ is higher than 2^{128} [10]. Thus, each seed (S_N) produces different $Major\ CA$ providing us with the huge possibility of 2^{128} different $Major\ CA$. This ensures that each key value (K) will encrypt differently and no key will be superfluous. Thus the CAC satisfies one of the important criterion of a secure cryptosystem. The criterion is specified by the following theorem:

Theorem: [11] A necessary condition for a cryptosystem to have a perfect secrecy is that it to have at least as many keys as messages.

Security Level 3 – Non-affine Transformation: This is a non-affine reversible CA transform which enables CAC to generate a non-affine group which is the *alternating group.* The affine group is a small subgroup of the alternating group(*Fig. 5*). The analytical proof that the CAC scheme generates alternating

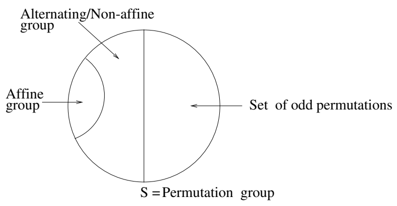


Fig. 5. Description of the permutation group

group is quite exhaustive and so omitted for short of space. Thus CAC, being able to generate the alternating group which is much larger than the affine group, satisfies another important criterion of a secure cryptosystem which says that ability to generate the alternating group on the message space is one of the strongest security conditions.

Security Level 4 – Key-mixing: The intermediate token ($\mathcal{T}_3$) is next XORed with the state S_N of the minor CA. This is very simple and takes only a single clock cycle. But it makes the encrypted token ($\mathcal{T}_{encr}$) totally unpredictable. The only way to return back to the original string is to randomly try with 128 bits which will cost $O(2^{128})$ operations, for every token.

Security Level 5: In order to further increase the level of security, our scheme can be used in bricklaying mode which will use multilevel encryption. This can be done with a minor increment of the cost while using the same basic structure reported in *Fig. 4*.

4.2 Cryptanalysis of *CAC*

The acceptance of any cryptosystem depends on its sustainability against various cryptanalysis attacks. Most important cryptanalysis are differential cryptanalysis [1] and Shannon's notion of perfect secrecy test [12]. We perform both these tests on CAC and as well as on DES and AES for the sake of comparison.

Results of Differential Cryptanalysis: We perform differential cryptanalysis with 50 different files having 11 different size. For each file, we take different fixed input differences to get the output probability distributions and the average value of the standard deviations for them is calculated. We also perform the same for DES and AES systems. The results are reported in *Table 1*. *Column II* of *Table 1* depicts the average mean standard deviation for CAC, where the same for DES and AES noted in *Column III* and *Column IV* respectively. The results for the current version of CAC is significantly better than that of DES and comparable to AES. It can also be noted from the results that the percentage of the standard deviation is around 4.0 whereas 10% is sufficient for a system to be considered as secured.

Results for Shannon's Security Quotient: We perform the Shannon's security test on CAC with 50 files for 9 different size and also perform the same on other cryptosystems (DES, AES) for the sake of comparison. *Column II* of

Table 1. Differential Cryptanalysis of our scheme and Comparative Study with DES and AES

Input file size (MB)	*Avg. Std. Devi*^{n} of XOR distributions for CAC (%)	*Avg. Std. Devi*^{n} of XOR distributions for DES (%)	*Avg. Std. Devi*^{n} of XOR distributions for AES (%)
1	4.36	31.95	4.2
2	4.30	30.03	4.0
4	4.26	29.05	3.63
6	4.17	28.24	3.62
8	3.91	28.10	3.67
10	4.02	28.89	3.52
12	3.89	28.74	3.51
14	3.55	28.52	3.48
16	3.40	27.86	3.43
18	3.42	27.74	3.26
20	3.59	27.67	3.24

Table 2. Measurement of Shannon's Security Quotient and comparative study with DES and AES

Input file size (MB)	Shannon's Security Quotient (ϑ) of CAC(%)	Shannon's Security Quotient (ϑ) for DES(%)	Shannon's Security Quotient (ϑ) for AES(%)
2	14.1605	14.2374	14.2345
3	11.5527	11.5531	11.5706
4	10.1060	10.2507	10.1675
7	7.5640	7.9141	7.6014
8	7.1182	7.1468	7.7046
9	6.7043	6.7139	6.7136
13	5.5868	5.5645	6.0266
14	5.3636	5.4001	5.4625
15	5.2097	5.3157	5.5552

Table 2 gives the average value of Security Quotient for our scheme calculated for different keys on each file size. The results show that our scheme fulfills the primary security level defined for any secure cryptosystem. *Column III and IV* of *Table 2* report the Security Quotient for DES and AES respectively, which establishes that CAC is better than DES and AES as far as Shannon's security notion is concerned.

4.3 Execution Time of Software Version

The main attractive feature of our CA based encryption scheme is its high speed of operation. Cellular automata are inherently parallel, so higher speed of execution of CAC is a natural outcome.

We have developed non-optimized reference code for CAC. Both the CAC and AES are run under same environment of $P-III$, $633MHz$ processor to

Table 3. Comparison of time of software version of *CAC* and *AES*

Input file size (in *MB*)	*CAC* Reference Code(in Sec)	*AES* Reference Code(in Sec)	*AES* Optimized Code(in Sec)
1.00	2.70	10.00	0.87
2.00	5.00	25.20	0.89
3.00	7.00	36.40	1.90
4.24	9.80	42.36	2.25
5.14	11.00	56.78	2.79
6.108	11.30	59.34	3.2
7.125	16.00	79.86	3.4
8.00	17.91	87.10	3.9
9.76	23.30	116.67	4.0
10.30	23.7	121.53	5.11
11.40	27.40	136.40	5.20
12.00	27.90	140.21	5.4

generate the results of *Table 3*. *CAC* reference code can be found to be significantly faster than that of *AES*. The optimized *CAC* code for commercial application is being developed. Preliminary results indicate that the optimized code of *CAC* will be faster than that of *AES*. However, the main advantage of *CAC* can be derived from its hardware version which is presented in the next section.

5 Cryto-Hardware Based On *CAC*

The pipelined architecture of *CAC* hardware is shown in *Figure 6*. The data path has five stages as explained below:

Stage 1 – Minor CA Implementation: It accepts the input key and control signals from *Control Block* and *CA Synthesis Hardware* block for on the fly generation of minor *CA*.

Stage 2 – Barrel Shifter Implementation: Its input register accepts the plain text token. The shift control of Barrel Shifter comes from *Stage 1*.

Stage 3 – Major CA Implementation with 3 Sub-Blocks: The three sub-blocks of this stage are flip-flops, a set of switches to implement Programmable *CA*(*PCA*) and an array of *XOR* gates. The control of the *PCA* to generate different major *CA* comes from the *CA* synthesis hardware.

Stage 4: It covers the implementation of *CMN*(Control Majority Not) logic along with evaluation of majority function on the pseudo exhaustive fields of major *CA*.

Stage 5 – The XOR operation: The input to this stage is the token coming from Stage 3 and the minor *CA* state.

Two inter-stage pipeline registers are introduced between Stages 3 and 4, and also between 4 and 5. Different features of *CAC* crypto-hardware are next reported:

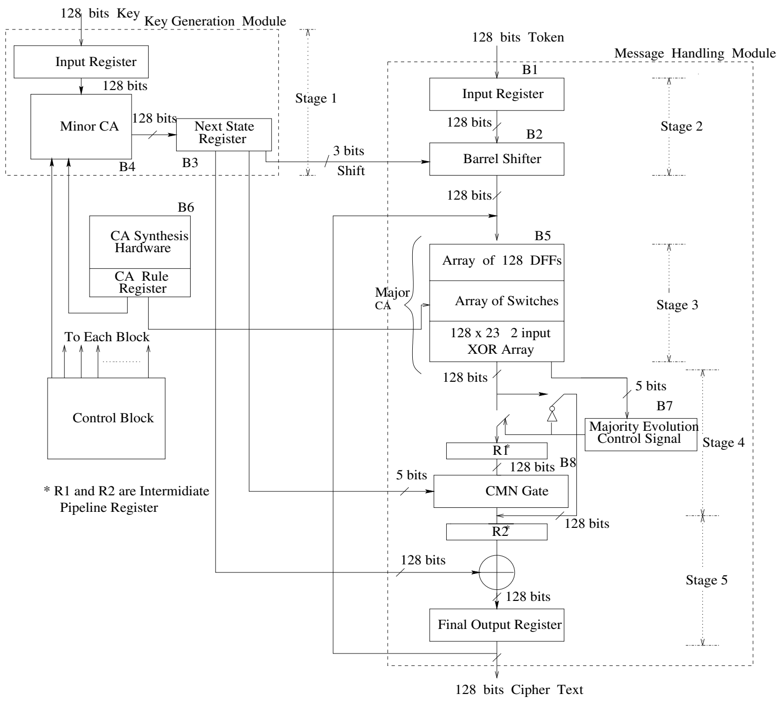


Fig. 6. Block Diagram *CAC* Hardware

- A verilog code has been written for the design and simulated using Cadence Verilog Simulator on a Sun Ultra-60 machine.
- The design has been synthesized and analyzed using Synopsis Design Compiler and Signal Scan.
- The design has been implemented with $0.25\mu CMOS$ technology.
- The pre-layout area estimate of the non-optimized design is 4.25×10^6 sq. micron.
- Static timing analysis of one complete run of *CAC* implementation on 128 bit plain text confirms correct operation of each stage with 1 GHz clock.
- For multiple rounds of operation(for the current implementation it is 4), the pipe line stages gets extended 4 times.
- The pipelined crypto-hardware throughput as per above timing analysis is 128 Gb/sec.
  Note: (i) Even if we assume 25% reduction of throughput for delay associated with silicon implementation, the throughput will be close to 100 Gb/sec. (ii) By contrast the full round Rijndael chip produced by NSA(National Security Agency) on $0.5CMOS$ *technology exhibit throughput of 5.7 Gb/sec. This is much lesser than the throughput of full round CAC chip.*
- Key generation hardware has been integrated within *CAC* implementation.

6 Conclusion

The *CA* based cryptosystem presented in this paper shows a very low cost, high speed encryption scheme with very high cracking complexity. The different cryptanalytic tests on our scheme shows that it satisfies primary security criterion and better than *DES*, *AES*. Its throughput is better than that of *AES*. The hardware version of *CAC* suits ideally for real time applications.

References

1. E. Biham and A. Shamir, "Differential cryptanalysis of des-like cryptosystems," *Journal of Cryptology*, vol. 4, pp. 3–72, 1991.
2. S. Wolfram, "Cryptography with cellular automata," *Proceedings of Crypto*, pp. 429–432, 1985.
3. P. Guan, "Cellular automaton public-key cryptosystem," *Complex System*, vol. 1, pp. 51–57, 1987.
4. H. Gutowitz, "Cryptography with dynamical systems," *ESPCI*, 1995.
5. S. Nandi, B.K. Kar, P.P. Chaudhuri "Theory and Application of Cellular Automata in Cryptography," *IEEE Trans. Computers, Vol. 43*, Dec. 1994.
6. S. R. Blackburn, S. Murphy, K. G. P. I. S. Group, and R. Holloway, "Comments on 'theory and applications of cellular automata in cryptography'," *IEEE Transaction on Computers*, 1999.
7. N. Ganguly, A. Das, B. K. Sikdar, and P. P. Chaudhuri, "Cellular automata model for cryptosystem," *Cellular Automata Conference, Yokohama National University, Japan*, 2000.
8. P. P. Chaudhuri and et. al., "Additive cellular automata theory and applications, vol 1" *IEEE Computer Society Press, California, USA*, 1997.
9. T. R. N. Rao and E. Fujiwara, "Error-control coding for computer systems," *Prentice-Hall, Englewood Cliffs, N.J.*
10. N. Ganguly, "Cellular automata evolution - theory and applications," *PhD Thesis (proposed), Bengal Engineering College, (D U), Howrah, India*, 2002.
11. D. Welsh, "Codes and cryptography," *Clarendon Press, Oxford*, 1988.
12. C. E. Shannon, "Communication theory of secrecy systems," *Bell Systems Technical Journal*, vol. 28, pp. 656–715, 1949.
13. B. Schneier, "Applied cryptography," *Second Edition, John Wiley & Sons*, 1996.

New Weak-Key Classes of IDEA

Alex Biryukov[⋆], Jorge Nakahara Jr[⋆⋆], Bart Preneel, and Joos Vandewalle

Katholieke Universiteit Leuven, Dept. ESAT/COSIC, Belgium
{abiryuko,nakahara,preneel,vdwalle}@esat.kuleuven.ac.be

Abstract. This paper presents a large collection of new weak-key classes for the IDEA cipher. The classes presented in this paper contain $2^{53} - 2^{64}$ weak keys (as compared with 2^{51} differential weak keys presented by Daemen at CRYPTO'93 and 2^{63} differential-linear weak-keys presented by Hawkes at EUROCRYPT'98). The novelty of our approach is in the use of boomerang distinguishers for the weak-key class membership test. We also show large weak-key classes for reduced-round versions of IDEA.

Keywords: IDEA cipher, weak keys, boomerang attack, NESSIE.

1 Introduction

The International Data Encryption Algorithm (IDEA) [8,9,10] is 64-bit block cipher using a 128-bit secret key. IDEA consists of eight rounds followed by an output transformation. In the last decade considerable cryptanalytic effort was concentrated on IDEA [1,3,4,5,6,7,11], however, despite that effort the cryptanalytic progress was very slow. Till now the best attack [1] breaks 4.5 rounds out of 8.5 rounds and it requires the knowledge of all 2^{64} blocks of the codebook and complexity of analysis is 2^{112}. In the same decade some weak-key classes for the full 8.5-round IDEA were found. In [4] a class of 2^{51} weak keys, detectable under differential membership test, was discovered. The membership test uses two chosen plaintexts and runs in at most 2^{12} steps. In [5] a class of 2^{63} weak keys, detectable under differential-linear membership test, was found.

In this paper we describe a series of new classes of weak-keys for the full 8.5-round IDEA. Keys are termed weak in the sense that some multiplicative keys which assume values 0 or 1 turn the modular multiplication into a linear operation. These key classes are detectable with boomerang techniques developed by Wagner ([13]). At least one of the weak-key classes is of size 2^{64} which is larger than the best previously known weak-key class of IDEA that used a differential-linear distinguisher. However, the complexity of the membership test for this class is 2^{16} data and time which is higher than for the Hawkes' class. We also

[⋆] The work described in this paper has been supported in part by the Commission of the European Communities through the IST Programme under Contract IST-1999-12324 and in part by the Concerted Research Action (GOA) project Mefisto 2000/06 of the Flemish Government.

[⋆⋆] Sponsored in part by the Concerted Research Action (GOA) project Mefisto 2000/06 of the Flemish Government.

show a collection of other smaller classes all of which are new and not covered by the previously known weak-key classes. In most cases, our membership test can be used to recover up to 16 additional key bits.

Furthermore, for 5-round IDEA (from the third to the seventh round), we found a class of 2^{95} weak keys with a boomerang membership test using only a single boomerang quartet (four chosen texts). This class can be extended four times to the class of size 2^{97} keys (a fraction 2^{-31} of all keys) at the cost of 2^8 quartets for the membership test (80% of success). The best class previously known for the 5-round IDEA contained 2^{13} times less keys [5]. This result also compares favorably to the currently best attack on 4.5-round IDEA mentioned above.

This paper is organized as follows. Section 2 gives a description of the IDEA block cipher, its key schedule and round structure. Section 3 describes the boomerang attack. Section 4 presents our discoveries of the new weak-key classes of IDEA. In Section 5 we show large fractions of weak keys for IDEA reduced to 5 rounds, and compare the complexities of some previously known attacks on IDEA. Section 6 contains a brief discussion and finally section 7 concludes the paper.

2 The IDEA Block Cipher

The International Data Encryption Algorithm (IDEA) is an iterated block cipher designed by Lai, Massey and Murphy in 1991 (see [8,9,10]).

In November 2000, IDEA was submitted as a candidate block cipher to the NESSIE Project [12], which is a project within the Information Societies Technology (IST) Programme of the European Commission.

The IDEA cipher has a 64-bit block size, 128-bit key size, and iterates eight rounds plus an output transformation. Three algebraic operations are used in IDEA: addition in $\mathbb{Z}_{2^{16}}$ denoted by $\boxplus$, bitwise exclusive-or, denoted by $\oplus$, and multiplication in $GF(2^{16}+1)$, denoted by $\odot$, with 2^{16} interpreted as 0. Encryption and decryption in IDEA use the same framework and differ only in the key schedule.

Every full round of IDEA can be split into two halves: a key-mixing layer and a multiplication-addition (MA) structure (see Fig. 1). Let $X^i = (X_1^i, X_2^i, X_3^i, X_4^i)$ be the input block to the i-th round of IDEA, where $1 \leq i \leq 8$, and $X_j^i \in \mathbb{Z}_2^{16}$, for $1 \leq j \leq 4$. Let $Z^i = (Z_1^{(i)}, Z_2^{(i)}, Z_3^{(i)}, Z_4^{(i)}, Z_5^{(i)}, Z_6^{(i)})$, with $Z_j^{(i)} \in \mathbb{Z}_2^{16}$, for $1 \leq j \leq 6$ represent the six subkey words used in the i-th round of IDEA. The first operation in a round is a key-mixing half-round that combines the four 16-bit input words with the subkey words $Z_1^{(i)}$, $Z_2^{(i)}$, $Z_3^{(i)}$, $Z_4^{(i)}$, in parallel, by either modular addition or multiplication. The result is input to the MA structure (or half-round), together with $Z_5^{(i)}$ and $Z_6^{(i)}$. At the end of the MA half-round there is a swap of the two middle words.

The output transformation (OT) is composed of a swap of the two middle input words and a key-mixing half-round.

2.1 Key Schedule of IDEA

The key schedule of IDEA processes the initial 128-bit master key into fifty-two 16-bit subkeys. Each one of the eight rounds uses six subkeys, and the output transformation (OT) uses four subkeys. The initial 128-bit key is partitioned into eight 16-bit words, and is used as the first eight subkeys. Successive sets of eight subkeys are generated by: rotating left by 25 bits the 128-bit block containing the previous eight 16-bit subkey words. Partitioning the resulting block into eight 16-bit words.

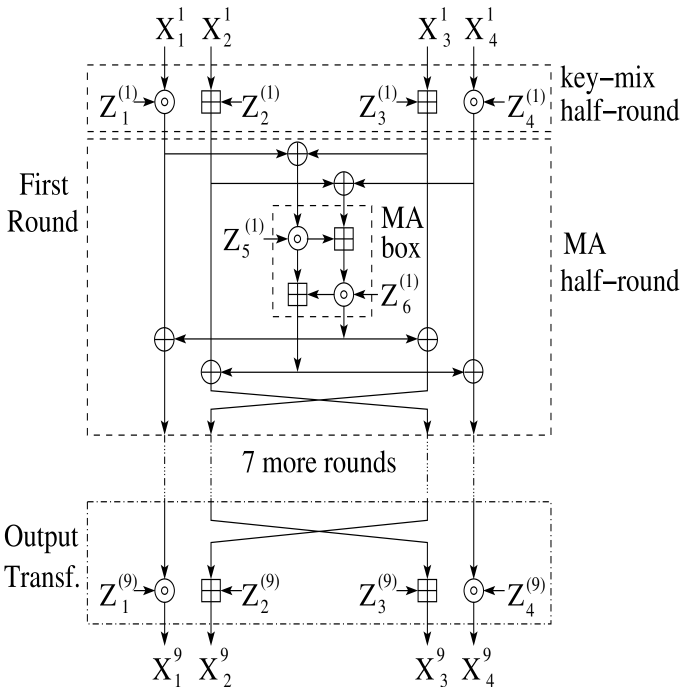

X_1^1 X_2^1 X_3^1 X_4^1
key–mix half–round
$Z_1^{(1)}$ $Z_2^{(1)}$ $Z_3^{(1)}$ $Z_4^{(1)}$
First Round
MA box
$Z_5^{(1)}$ $Z_6^{(1)}$
MA half–round
7 more rounds
Output Transf.
$Z_1^{(9)}$ $Z_2^{(9)}$ $Z_3^{(9)}$ $Z_4^{(9)}$
X_1^9 X_2^9 X_3^9 X_4^9

Fig. 1. Encryption scheme of IDEA block cipher.

Table 1 shows the dependency of subkey bits on the master key bits, which is indexed from 0 (MSB: most significant bit) to 127 (LSB: least significant bit). Bit numbering is taken modulo 128, that is, in a circular fashion, due to the rotation operation.

3 The Boomerang Attack

In this section we describe the cryptanalytic technique called the *boomerang attack* developed by Wagner [13].

Traditional differential attacks [2] are powerful methods of cryptanalysis in which the attacker considers pairs of plaintexts P_1, P_2 with a fixed difference

Table 1. Dependency of subkey bits on the master key bits of IDEA.

i-th round	$Z_1^{(i)}$	$Z_2^{(i)}$	$Z_3^{(i)}$	$Z_4^{(i)}$	$Z_5^{(i)}$	$Z_6^{(i)}$
1	0–15	16–31	32–47	48–63	64–79	80–95
2	96–111	112–127	25–40	41–56	57–72	73–88
3	89–104	105–120	121–8	9–24	50–65	66–81
4	82–97	98–113	114–1	2–17	18–33	34–49
5	75–90	91–106	107–122	123–10	11–26	27–42
6	43–58	59–74	100–115	116–3	4–19	20–35
7	36–51	52–67	68–83	84–99	125–12	13–28
8	29–44	45–60	61–76	77–92	93–108	109–124
OT	22–37	38–53	54–69	70–85	—	—

$\triangle = P_1 \oplus P_2$, and studies the propagation of differential patterns throughout the cipher. The aim of the attacker is to predict the resulting ciphertext difference $C_1 \oplus C_2$ with non-negligible probability. If this can be done then the cipher can be distinguished from a random permutation, and in many cases a key-recovery attack can be mounted on the cipher.

The boomerang attack is a differential-style attack in which the attacker does not try to cover the whole cipher with a single highly-probable differential pattern. Instead, the attacker tries to find several high-probability patterns that are not necessarily related to each other but together cover the whole cipher. The boomerang attack requires the ability to make both chosen-plaintext and chosen-ciphertext queries[1].

Let's denote the encryption operation by E and its decomposition into two parts (not necessarily dividing the cipher into halves) as $E = E_1 \circ E_0$. Suppose that we start with two plaintexts P_1, P_2, such that $P_1 \oplus P_2 = \triangle$. Suppose that we have a differential pattern $\triangle \to \triangle^*$ propagating through the E_0 part of the cipher with probability p. Now consider the corresponding ciphertexts C_1, C_2 and their "shift" by the difference ∇ as follows: $C_3 = C_1 \oplus \nabla$, $C_4 = C_2 \oplus \nabla$. As ∇ we use a pattern that goes up through E_1^{-1} with high probability q, i.e. $\nabla \to \nabla^*$. We decrypt the new ciphertexts C_3, C_4 to obtain their corresponding plaintexts P_3 and P_4. If the previous three difference patterns happened as predicted, between E_0 and E_1, we obtain:

$$E_0(P_3)\oplus E_0(P_4) = E_0(P_1)\oplus E_0(P_2)\oplus E_1^{-1}(C_1)\oplus E_1^{-1}(C_3)\oplus E_1^{-1}(C_2)\oplus E_1^{-1}(C_4) =$$

$$= \triangle^* \oplus \nabla^* \oplus \nabla^* = \triangle^*.$$

Thus we can decrypt backwards through E_0 using the pattern $\triangle^* \to \triangle$. The claim is that with probability p^2q^2, the difference $P_3 \oplus P_4 = \triangle$ holds, which can be readily checked. See Figure 2 for a graphical representation of a boomerang quartet. This is an example of a **top-down boomerang**.

[1] The chosen-plaintext queries in the boomerang attack are *adaptive* in the sense that one first obtains ciphertexts which are the results of the chosen-plaintext queries to the encryption oracle, then one performs appropriate modifications to these ciphertexts and finally feeds them to the decryption oracle.

There are several refinements to the technique described above: we may send boomerangs from the decryption direction, that is, starting from the ciphertext and then performing adaptive plaintext queries (**bottom-up boomerangs**); we may guess the keys of the first or last rounds and send boomerangs (top-down or bottom-up) based on the key guesses. We can use the careful choice of $\triangle$ and ∇ in order to obtain additional half-round(s) in the middle (for the IDEA cipher it is the MA half-round) for free (this observation was used in [13] to attack the Khufu-16 cipher). We can use truncated differentials whenever the two faces of the boomerang produce the same difference patterns, the boomerang goes through, no matter what these difference patterns are. If the E_0 part is short enough we may use more analysis to check how differences propagate through it, without waiting for the perfect match of the difference in the second pair of plaintexts to $\triangle$.

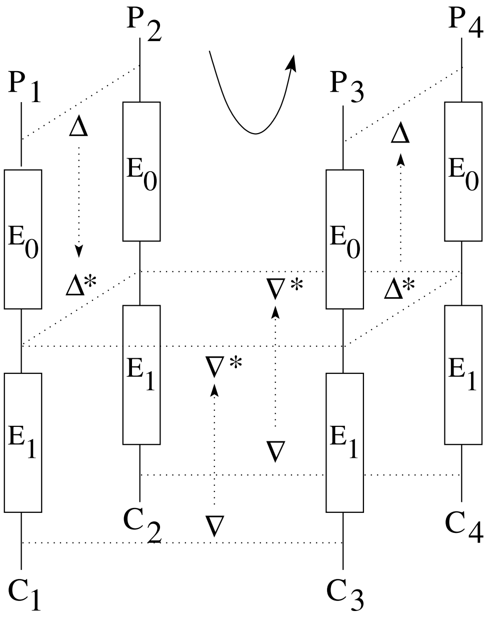


Fig. 2. A (top-down) boomerang quartet (P_1, P_2, C_3, C_4).

4 Boomerang Attack under Weak-Key Assumptions

In this section we show a variety of attacks on the full 8.5-round IDEA block cipher under some weak-key assumptions. These attacks provide new weak-key classes, larger than the ones discovered by Daemen [4], and not covered by the class discovered by Hawkes [5]. Some of these classes are the largest found so far for this cipher, but they require more effort for their membership test compared to Hawkes' differential-linear classes.

In order to build our new weak-key classes we use the boomerang-style distinguisher. The benefit of using boomerang distinguishers is two-fold: first, boomerang distinguishers pose different constraints on the key schedule than the previous differential or differential-linear distinguishers, thus we are likely to find new weak-key classes; second, we can pick unrelated patterns to cover both the E_0 (top) and E_1 (bottom) parts of the cipher and optimize the number of key-bit constraints to be minimal. As in the previous attacks we consider input xor-differences that only differ in the most significant bit (8000_x). Such differences have the advantage of propagating across the modular addition for free (i.e. with probability one). We are thus concerned only with the multiplicative keys.

4.1 Advanced Boomerang Techniques

In this section we describe our general method of search for the weak-key classes of the IDEA cipher. The method includes several refinements to the basic boomerang attack. These refinements help us to increase the key-class sizes.

We have written a program that searches through all possible plaintext/ciphertext differences in order to find the largest boomerang weak-key classes. We also considered gaps[2] of one, two and three half-rounds in the middle of the cipher, in order to increase the sizes of the key-classes at the cost of higher data and time complexities of the membership test. Another relaxation was not to cover either the top-most or bottom-most key-mixing half-round, assuming that the attacker can guess the required top or bottom keys or use special structures to construct appropriate input (or output) difference after the key-mixing. In these cases, given a correct boomerang quartet, the attacker can find up to 16 bits of multiplicative subkeys of the first or last key-mixing half-rounds, in addition to the zero key bits of the weak-key class. In the following subsections we describe several examples of our weak key classes together with their membership tests. In Table 2 we summarize the findings of this paper and compare them with the previously best classes.

4.2 A Weak-Key Class of Size 2^{53}

Consider a boomerang distinguisher which consists of two differentials: one with chosen-plaintext xor-difference $\triangle = (8000_x, 0000_x, 0000_x, 0000_x)$ that causes the xor-difference $\triangle^* = (8000_x, 8000_x, 0000_x, 8000_x)$, after 2.5 (encryption) rounds with probability one, provided that the 64 key bits 0–23, 64–103 are zero. The other differential has chosen-ciphertext xor-difference $\nabla = (0000_x, 8000_x, 8000_x, 0000_x)$, and causes the xor-difference $\nabla^* = (0000_x, 8000_x, 0000_x, 8000_x)$ after 5.5 (decryption) rounds with probability one, provided the 63 key bits numbered 0–25, 77–107, 123–127 are zero. These two differentials together require that the 75 key bits numbered 0–25, 64–107, 123–127 be zero. One MA half-round, with subkeys $Z_5^{(3)}$ and $Z_6^{(3)}$, is not included in the boomerang. However due to the proper choice of the differences coming from top and the bottom end of the cipher, we gain this MA half-round for free (a similar trick was used by Wagner

[2] Half-rounds with no constraints on the key bits.

in his attack on Khufu [13]). This boomerang can be used to identify a weak-key class of size $2^{128-75} = 2^{53}$ using a single quartet: two chosen-plaintext and two chosen-ciphertext queries.

4.3 A Weak-Key Class of Size 2^{56}

Consider a boomerang distinguisher which consists of two differentials: one with chosen plaintext xor-difference $\triangle = (8000_x, 0000_x, 8000_x, 0000_x)$ that causes the xor-difference $\triangle^* = (8000_x, 8000_x, 0000_x, 0000_x)$ after 1.5 (encryption) rounds with probability one, provided that the 30 key bits 0–14, 96–110 are zero. The other differential has chosen xor-difference $\nabla = (0000_x, 8000_x, 0000_x, 8000_x)$ at the end of the 8th round (without including the last key mix half-round) and causes the xor-difference $\nabla^* = (0000_x, 8000_x, 0000_x, 8000_x)$ at the beginning of the third round with probability one, provided the 42 key bits 15–32, 84–95, 116–127 are zero. These two differentials jointly require that the 72 key bits 0–32, 84–110, 116–127 be zero. One MA half-round is not included in the boomerang. If the fourteen MSBs of $Z_4^{(9)}$ can be guessed, then this **bottom-up boomerang** can be used to identify a weak-key class of size $2^{128-72} = 2^{56}$. However we can do better than guessing the last subkey. Instead, we will prepare two pools of ciphertexts which have appropriate differences in all words and random 2^9 values in the ciphertext word after the unknown key $Z_4^{(9)}$. The pools contain 2^{18} pairs and we assume that due to the birthday paradox we'll have several pairs that have the required difference 8000_x after the decryption by the unknown key. If this event happens, then the boomerang runs for the rest of the rounds with probability one (the MA half-round in the middle is bypassed for free). In order to detect this class we thus need 2^{11} queries.

4.4 The Largest Weak Key Class of Size 2^{64}

In this section we describe the largest weak-key class of the IDEA cipher which we discovered. It is twice larger than that described by Hawkes, although its membership test is more complex. We consider Daemen's weak-key class (see Table 2) in which we do not restrict the subkey $Z_4^{(7)}$ thus increasing the key class size from 51 to 66 bits. We use the boomerang distinguisher **bottom-up** and create special structures of ciphertexts in order to bypass the bottom round without the need to guess the multiplicative key $Z_1^{(8)}$.

In more detail: we produce two pools of chosen ciphertexts of size 2^{14} texts each, in order to generate many pairs with the difference $\triangle = (0000_x, 8000_x, 0000_x, 8000_x)$ just above the bottom key-mixing half-round. This pattern covers the next MA half-round with probability one, since it causes zero difference in the inputs to the MA-box. We create a pool C_1 of 2^{16} chosen ciphertexts, in which the 4th word takes 2^{14} random values and the other three words are arbitrary but fixed for all the texts in the pool. The second pool C_2 is created using the difference $(0000_x, 8000_x, 0000_x)$ from the texts of the first pool in the 1st, 2nd and 3rd words respectively, and the 4th word runs through random 2^{14} values. Thus, between the pools we have 2^{28} pairs with the difference $(0000_x, 8000_x, 0000_x, *)$ and among these we have 2^{12} pairs with the required difference (0000_x, 8000_x, 0000_x, 8000_x) after decrypting a single key-mixing half-round and thus with a difference (0000_x, 8000_x, 0000_x, 8000_x) after decrypting the last round.

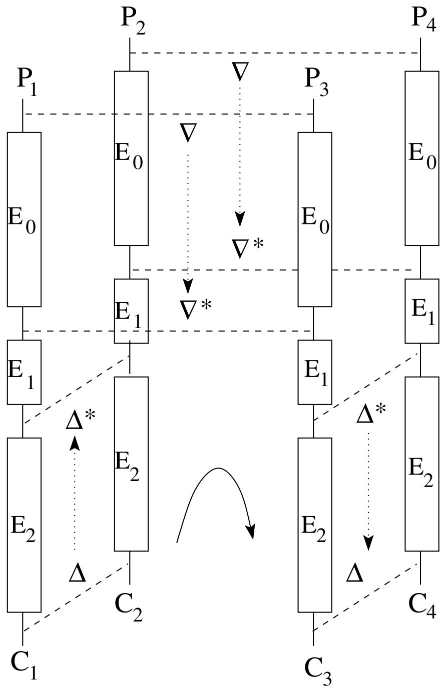


Fig. 3. Boomerang quartet for our largest key-class. E_0: 6 rounds, E_1: 1.5 rounds, E_2: 1 round.

We decrypt the pools C_1, C_2 producing plaintext pools P_1, P_2, from which we create two new pools P_3, P_4 by using the difference ∇ = (0000_x, 8000_x, 0000_x, 8000_x) and encrypt these pools to obtain new ciphertext pools C_3, C_4. We sort these pools by the highest 48 bits (1st, 2nd and 3rd words) and check if there is a pair with difference: $0000_x, 8000_x, 0000_x$ in the highest-order 48 bits, between the pools. If so, we proclaim that the boomerang has returned and the key belongs to the weak-key class.

In this boomerang attack we bypass the bottom key-mixing half-round using the birthday paradox, and we cover the IDEA cipher with one round from the bottom and six rounds from the top. In between we have a gap of three half-rounds: key-mixing, MA and another key-mixing which our differences have to bridge in order for the boomerang to work. Some of the multiplicative keys in the gap are already restricted by our key class which helps for the boomerang differences to bridge the gap. Although the key mask for this class has 66 bits we observed that boomerangs returned for about 25% of all such keys, for 2^{14}

quartets, which reduces the key class to 64 bits[3]. Data complexity of this membership test is 2^{14} quartets or 2^{16} texts. Figure 3 shows one quartet used for our largest key class.

5 Attacks on Round-Reduced IDEA

In this section we show boomerang attacks on round-reduced IDEA under weak-key assumptions.

5.1 Attack on 5-Round IDEA

Using the program described above we discovered that 5-round IDEA (from the 3rd to the 7th round) can be attacked for a fraction of 2^{-33} keys (2^{95} weak keys) with just one quartet (4 text queries). The plaintext difference to the boomerang is $\triangle = (0000_x, 8000_x, 0000_x, 8000_x)$ and the ciphertext difference after 5 rounds is $\nabla = (8000_x, 8000_x, 0000_x, 8000_x)$. The key-mixing at the 10th half-round is covered for free.

There is another smaller class with 2^{92} weak keys (fraction of 2^{-36}) also from the 3rd to the 7th round which can be detected with a single boomerang quartet. Compare these results to the class of size 2^{84} (fraction 2^{-44}) found previously by Hawkes. Recall also that the best attack on IDEA [1] covers only 4.5 half-rounds, uses all the 2^{64} blocks of the codebook and has 2^{112} complexity.

Furthermore there is a larger class of size 2^{97} (fraction 2^{-31}) which requires 2^8 quartets for 80% success probability of the boomerang attack. The increased data requirements are due to the two half-round gap in the middle of the boomerang. Also note that this class includes the previous class of size 2^{95}.

In Table 2 we summarize our results for round-reduced versions of IDEA (from four to six rounds) and compare them to the best-previously known results [5]. In this table "Flow" indicates the top-down ($\downarrow$), or bottom-up ($\uparrow$) direction of the boomerang attack, $|WKC|$ denotes the size of the weak key class. The data complexity is measured in the number of texts (divide by four to get the number of quartets), time complexity is measured in the number of reduced-round encryptions.

6 Discussion

We have also discovered that not only keys with certain subkeys equal to zero or one are weak (as was known before) but keys with few runs of ones are also weak and contribute to a very slow avalanche inside the IDEA cipher. While the zero-one weak keys problem of IDEA can be corrected just by XORing a fixed constant to all the keys (one such constant may be `0DAE`$_\mathtt{x}$ as suggested in [4]) the problem with the runs of ones may still remain and will require complete redesign of the IDEA key schedule.

[3] By increasing the number of quartets more keys would be covered but the amount of additional data required are larger than the gain in the key bits.

Table 2. Summary of Weak-key Boomerang Distinguishers.

#Rnds	Hawkes $\|WKC\|$	Our $\|WKC\|$	Half-Rnds♯	Flow	Weak-Key Bit Positions	Input‡ Difference	Output‡ Difference	Complexity Data	Complexity Time
4	2^{99}	2^{104} ♭	6-13	↓	11-32	(0 0 1 1)	(1 1 0 1)	2^{10}	2^{10}
4.5	2^{97}	2^{101}	6-14	↑	0-18, 123-127	(0 1 0 1)	(0 1 0 1)	2^{18}	2^{18}
4.5	2^{97}	2^{101}	4-12	↑	2-25	(0 1 0 1)	(0 1 0 1)	2^{18}	2^{18}
5	2^{84}	2^{97}	2-11	↓	0-25, 123-127	(0 1 0 *)	(0 1 0 1)	2^{10}	2^{10}
5	2^{84}	2^{97}	4-13	↓	0-18, 116-127	(0 1 0 *)	(0 1 0 1)	2^{10}	2^{10}
5	2^{84}	2^{95}	4-13	↓	2-34	(0 1 0 1)	(1 1 0 1)	4	4
5	2^{84}	2^{97}	4-13	↓	2-32	(0 1 0 1)	(1 1 0 1)	2^{8}	2^{8}
5.5	2^{82}	2^{95}	4-14	↑	2-34	(0 1 0 1)	(* 1 0 *)	2^{16}	2^{16}
5.5	2^{82}	2^{97}	4-14	↑	2-32	(0 1 0 1)	(1 1 0 1)	2^{22}	2^{22}
6	2^{82}	2^{83}	2-13	↓	0-32, 116-127	(0 1 0 *)	(0 1 0 1)	2^{10}	2^{10}
8.5	2^{63}	2^{53}	0-16	↓	0-25,64-107,123-127	(1 0 0 0)	(0 1 1 0)	4	4
8.5	2^{63}	2^{56}	0-16	↑	0-32,84-110,116-127	(1 0 1 0)	(0 0 1 *)	2^{11}	2^{11}
8.5	2^{63}	2^{57}	0-16	↓	0-23,57-91,116-127	(0 0 1 *)	(0 0 1 1)	2^{11}	2^{11}
8.5	2^{63}	2^{57}	0-16	↓	0-32,57-91,125-127	(0 0 1 *)	(0 0 1 1)	2^{13}	2^{13}
8.5	2^{63}	2^{58}	0-16	↑	0-18,41-71,77-91,123-127	(0 1 0 1)	(0 1 0 *)	2^{24}	2^{24}
8.5	2^{63}	2^{59}	0-16	↑	0-32,84-107,116-127	(1 0 1 0)	(0 0 1 *)	2^{21}	2^{21}
8.5	2^{63}	2^{59}	0-16	↓	0-25,73-110,123-127	(0 0 0 *)	(0 1 1 0)	2^{16}	2^{16}
8.5	2^{63}	2^{60}	0-16	↓	4-25,66-110	(0 0 0 *)	(0 1 1 0)	2^{24}	2^{24}
8.5	2^{63}	2^{64}	0-16	↑	0-25,77-107,123-127	(0 1 0 1)	(0 1 1 *)	2^{16}	2^{16}

‡: the symbol '0' denotes 32-bit difference `0000x`, and '1' denotes `8000x`; '*' denotes arbitrary difference, used to produce `8000x` difference after multiplication by an unrestricted key.
♯: half-round numbering starts from 0, and ends at 16.
♭: out of a class of size 2^{106}, about 1/4 of the keys allows boomerangs to return, for the given amount of data.

7 Conclusions

IDEA is a strong block cipher that for more than a decade has evaded attempts of cryptanalysis. However during the same period of time large weak key classes for this cipher were found. This is due to the fact that the main non-linear part of the cipher is based on multiplication with a chosen master key and due to linearity of the key schedule. In this paper we have shown new large weak-key classes of IDEA. We have used the boomerang distinguisher as a membership test for these classes. These results strengthen the need for the redesign of the key schedule of IDEA.

References

1. Biham, E., Biryukov, A., Shamir, A.: Miss-in-the-Middle Attacks on IDEA, Khufu and Khafre, *6th Fast Software Encryption Workshop, LNCS 1636*, L.R. Knudsen, Ed., Springer-Verlag, 1999, 124–138.
2. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard, Springer-Verlag, 1993.
3. Borst, J., Knudsen, L.R., Rijmen, V.: Two Attacks on Reduced IDEA (extended abstract), *Advances in Cryptology, Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, 1–13.

4. Daemen, J., Govaerts, R., Vandewalle, J.: Weak Keys for IDEA, *Advances in Cryptology, Crypto'93, LNCS 773*, D.R. Stinson, Ed., Springer-Verlag, 1994, 224–231.
5. Hawkes, P.: Differential-Linear Weak Key Classes of IDEA, *Advances in Cryptology, Eurocrypt'98, LNCS 1403*, K. Nyberg, Ed., Springer-Verlag, 1998, 112–126.
6. Hawkes, P., O'Connor, L.: On Applying Linear Cryptanalysis to IDEA, *Advances in Cryptology, Asiacrypt'96, LNCS 1163*, K. Kim and T. Matsumoto, Eds., Springer-Verlag, 1996, 105–115.
7. Kelsey, J., Schneier, B., Wagner, D.: Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER and Triple-DES, *Advances in Cryptology, Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, 237–251.
8. Lai, X.: On the Design and Security of Block Ciphers, Hartung-Gorre Verlag, Konstanz, 1992.
9. Lai, X., Massey,J.L.: A Proposal for a New Block Encryption Standard, *Advances in Cryptology, Eurocrypt'90, LNCS 473*, I.B. Damgård, Ed., Springer-Verlag, 1990, 389–404.
10. Lai, X., Massey, J.L., Murphy, S.: Markov Ciphers and Differential Cryptanalysis, *Advances in Cryptology, Eurocrypt'91, LNCS 547*, D.W. Davies, Ed., Springer-Verlag, 1991, 17–38.
11. Meier, W.: On the Security of the IDEA Block Cipher, *Advances in Cryptology, Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, 371–385.
12. *NESSIE Project – New European Schemes for Signatures, Integrity and Encryption* – available at `http://cryptonessie.org`.
13. Wagner, D.: The Boomerang Attack, *6th Fast Software Encryption Workshop, LNCS 1636*, L.R. Knudsen, Ed., Springer-Verlag, 1999, 156–170.

A Example of 5-Round Boomerang

In this section we show an example printout of our program for the 5-round boomerang from the 3rd to the 7th rounds (4th – 13th half rounds). The differences are the input differences to the half-rounds, the keys printed are those that need to be restricted and the masks show the effect of these restrictions on the weak-key class size (denoted by $|WKC|$). The key-class below has size 2^{95}. The gap in half-round 10 is covered for free.

```
 HR     xor-difference     weak subkeys   key masks                 |WKC|
----|-------------------|---------------|--------------------------|-------
  4  0000 8000 0000 8000  Z_4^3 ff8000ff fffffff fffffff fffffff  2^113
  5  0000 8000 0000 8000        ff8000ff fffffff fffffff fffffff  2^113
  6  0000 0000 8000 8000  Z_4^4 c00000ff fffffff fffffff fffffff  2^106
  7  0000 0000 8000 8000  Z_5^4 c0000000 7ffffff fffffff fffffff  2^97
  8  0000 8000 8000 0000        c0000000 7ffffff fffffff fffffff  2^97
  9  0000 8000 8000 0000  Z_5^5 c0000000 7ffffff fffffff fffffff  2^97
 10  ---- ---- ---- ----
 11  8000 0000 0000 0000  Z_5^6 Z_6^6 c0000000 1ffffff fffffff fffffff 2^95
 12  0000 8000 0000 0000        c0000000 1ffffff fffffff fffffff  2^95
 13  0000 8000 0000 0000  Z_6^7 c0000000 1ffffff fffffff fffffff  2^95
```

Below we show another example of a weak-key class. This class is of size 2^{97} and includes the class shown above. However due to the gap of 2 half-rounds (10th and 11th) the membership test for this class requires 2^8 quartets for 80% success probability.

```
 HR     xor-difference     weak subkeys   key masks                  |WKC|
----|-------------------|---------------|--------------------------|-------
  4  0000 8000 0000 8000  Z_4^3 ff8000ff ffffffff ffffffff ffffffff  2^113
  5  0000 8000 0000 8000        ff8000ff ffffffff ffffffff ffffffff  2^113
  6  0000 0000 8000 8000  Z_4^4 c00000ff ffffffff ffffffff ffffffff  2^106
  7  0000 0000 8000 8000  Z_5^4 c0000000 7fffffff ffffffff ffffffff  2^97
  8  0000 8000 8000 0000        c0000000 7fffffff ffffffff ffffffff  2^97
  9  0000 8000 8000 0000  Z_5^5 c0000000 7fffffff ffffffff ffffffff  2^97
 10  ---- ---- ---- ----
 11  ---- ---- ---- ----
 12  0000 8000 0000 0000        c0000000 7fffffff ffffffff ffffffff  2^97
 13  0000 8000 0000 0000  Z_6^7 c0000000 7fffffff ffffffff ffffffff  2^97
```

Risks with Raw-Key Masking – The Security Evaluation of *2-Key XCBC*

Soichi Furuya[1] and Kouichi Sakurai[2]

[1] Systems Development Lab., Hitachi, Ltd.
soichi@sdl.hitachi.co.jp
[2] Graduate School of I.S.E.E., Kyushu University
sakurai@csce.kyushu-u.ac.jp

Abstract. There are extensive researches on how CBC-MAC can be modified in order to efficiently deal with messages of arbitrary lengths. Based on the three-key construction of XCBC by Black and Rogaway, Moriai and Imai improved the scheme and proposed an optimally efficient CBC-MAC variants with two key materials, that is called 2-key XCBC. They give a proof of the security in the same manner as 3-key XCBC. In this paper, we study 2-key XCBC, and discuss the security of 2-key XCBC used with real replacement to an ideal PRP. We show (1) a forgery based on the *raw-key masking* technique used in 2-key XCBC for a particular instance where Even-Mansour PRP construction is used, and (2) an attack that violates the provable security of DESX construction. Therefore, the raw-key masking technique, which is the core improvement of 2-key CBC, must be avoided unless an overall implementation is considered in detail. Moreover, we discuss 2-key XCBC with two promising real block ciphers AES and Camellia and note important security consideration concerning their uses with 2-key XCBC.

Keywords: Block cipher, mode of operation, MAC, provable security.

1 Introduction

CBC-MAC [FIPS81] is the most commonly used way to generate a message authentication code (MAC) from a block cipher. This was originally proposed as a mode of operation for DES cipher [FIPS46-3]. Assuming that CBC-MAC is used with a pseudorandom permutation (PRP), the MAC tag cannot be forged by an attacker that does not change the length of message [BKR94]. On the other hand, it is also well-known that the basic CBC-MAC can be forged if an attacker changes the length of verified messages.

To overcome this problem, some improvements of CBC-MAC are proposed. Major objectives of the improvements are followings:

1. a secure MAC generation even against concatenating attacks,
2. dealing with a message of arbitrary length, (i.e., message can be authenticated regardless of the block size of a block cipher),
3. minimum PRP invocation,
4. minimum size of key material.

In order to achieve the security against concatenation attacks, the scheme needs the distinct terminating process. One of the solutions against this problem is proposed as EMAC at RACE project [RACE]. EMAC always requires two key setups for one message. Moreover, in order to handle a message that does not fit the block boundary, the message must be padded such that any length of bit strings are transformed to non-zero multiple length of blocks. Consequently, another yet PRP invocation is required for those messages that fit the block boundary. In order to minimize these additional costs Black and Rogaway constructed an efficient CBC-based MAC schemes. The most successful construction is called as 3-key XCBC[BR00]. This is initiated by the important lemma in the literature [BR00]. This lemma gives a way to generate two computationally different PRPs out of one secret PRP, and proves the security of the proposed construction. 3-key XCBC requires no additional PRP invocation or no more than one key setups, but requires another $2b$-bit independent key materials where b is the block length of the PRP.

Moriai and Imai enhanced the lemma shown by Black and Rogaway. The enhanced lemma contribute to reduce the length of the necessary key material. Moriai and Imai proposed an enhanced MAC scheme, that is called as 2-key XCBC [MI02a]. 2-key XCBC achieves most of the desirable features of CBC based MAC schemes. From this point, 2-key XCBC is the most efficient MAC scheme amongst ones based on the CBC-MAC.

In this paper, we study the security of 2-key XCBC with real ciphers and discuss the gap between real ciphers and a PRP assumed in the security proof of 2-key XCBC. We firstly consider the security of 2-key XCBC with two provable secure PRP constructions. One is Even-Mansour construction [EM97] and give the way to forge example schemes of 2-key XCBC mode with EM construction. Next, DESX by Rivest [KR96] is considered. Although our modified attack does not efficiently forge 2-key XCBC with DESX, the computational complexity required for our attack is much below than the proven boundary for DESX.

Related works. There are also extensive research results dealing with a pseudorandom function (PRF) and a pseudorandom permutation (PRP). The definition of PRF and PRP are given in [GGM86]. Luby and Rackoff analyzed the way to use PRF in order to construct a PRP [LR88]. Major modes of operation are analyzed using PRP/PRF and their prooves of security appears in [BD+97]. For the practical security assessment of Even-Mansour construction, Daemen gives two kinds of attacks [Da93]. Although they are faster than the key exhaustive search, the computational complexity required for the attack does not conflict the provable security of EM construction shown in [EM97]. The attacks described by Daemen are essentially the partial key exhaustive search and a time-data tradeoff variant. In addition the advanced slide attacks give more efficient attack to derive the key of EM construction [BW00]. For the security of DESX construction. The proof of the security is given in [KR96]. On the other hand there are some results on the attack of DESX. The advanced slide attack is again applicable to DESX [BW00]. There is another unpublished note on the security of DESX [Mi02]. For the provable secure MAC schemes based on a PRP, there is a recent publication [BR02]. The proposed scheme PMAC is as efficient as CBC-MAC. In addition to CBC-MAC PMAC has the provable security and potential parallelizability. There is a result on the strengthened CBC-MAC in terms of provable security [JJV02]. More recently, another CBC-based MAC scheme was proposed for NIST [KI02].

2 Notations

For two integers a and b, the relation $a \mid b$ means that a divides b. For two bit strings, a and b, the operation $a \oplus b$ means the bitwise xor operation. The operation $a \lll_n k$ means the k-bit left rotation in an n-bit register, i.e. a is treated as an n-bit string if a is shorter than that.

In the following part, we write $P_K(M)$ as a PRP with two inputs, a key K and a message data M. Let b the block size of $P_K(M)$ in bit length. Let $len(M)$ be the bit length of M. More specifically we use the notation $\#_b M$ to describe the number of b-bit blocks required to store M. Therefore, $\#_b M = \lceil len(M)/b \rceil$. For a bit string M such that $b \mid len(M)$, we use the notation with a subscript M_i to be the i-th block of M cutting into b-bit blocks. For two bit strings A and B, the operation $A \mid\mid B$ means the concatenation of two strings.

3 MAC Generations Based on a CBC-MAC

So far, some MAC generating schemes have been proposed, based on the original CBC-MAC[FIPS81]. We review some relative works to our research.

3.1 CBC-MAC and Variants

The CBC-MAC operates on a message M such that $b \mid m$ and $len(M)\ /= 0$. The authentication tag tag_M is generated in the following way:

$$tag_M = CBCMAC_K(M) = P_K(M_{\#M_i - 1} \oplus P_K(M_{\#M_i - 2} \oplus \cdots \oplus P_K(M_0))).$$

Assuming that P is a PRP, an adversary without a secret key K cannot forge a message when an adversary does not change the length of the message [BKR94]. However if an adversary changes the length of a message, the adversary can generate a different message with a valid tag only out of a valid pair (M, tag_M) as follows:

$$CBCMAC_K(M \mid\mid M' \mid\mid M' \mid\mid \ldots \mid\mid M') = tag_M,$$

where M' is a string to be generated out of M by replacing M_0 by $M_0 \oplus tag_M$. We call this attack as the concatenation attack.

Following the analysis on the original CBC-MAC, some schemes being secure against the concatenation attack were proposed. One of the examples is

ECBC [RACE]. To overcome the drawbacks of ECBC (dual key setups, additional PRP invocations, handling arbitrary length of messages), some constructions with three key materials are introduced. The technical essence of the three key construction is that for a message such that $len(M) > 0$ and $b \mid len(M)$, the obligative padding is not applied (in order to avoid an additional P invocation), but another PRP is applied. To realize the idea, following methods are proposed:

1. Another key material K_3 is introduced so as to identify messages with or without padding by using different PRPs, one is P_{K_2}, and the other P_{K_3}.
2. Unlike EMAC, the last invocation of CBC is the target of switching two PRPs.

The 3-key ECBC proposed in [BR00] is the method with (1) and the 3-key FCBC proposed in [BR00] is the method with (1) and (2). The FCBC achieves the minimum P invocation for any message. Meanwhile the FCBC requires three keys all of which are set to P [1]. In real implementation of the FCBC scheme, three keys may be fed into a key schedule that can be costful when dealing shorter messages.

In [BR00], Black and Rogaway introduced an almost optimally efficient scheme based on CBC-MAC. The number of key setups for the scheme is reduced to one instead of three (FCBC). There is a most contributing lemma to this improvement. The lemma, Lemma 4 in [BR00], proves that there is no adversary that efficiently distinguishes a pair of two functions, $(P_{K_1}(\cdot), P_{K_2}(\cdot))$ and $(P_{K_1}(K_2 \oplus \cdot), P_{K_1}(\cdot))$. Thanks to the lemma, two permutations P_{K_2} and P_{K_3} of FCBC can be replaced by two functions $P_{K_1}(\cdot \oplus K_2)$ and $P_{K_1}(\cdot \oplus K_3)$ without security degrade. While the provable security is still achieved, the number of key setups became optimal.

3.2 2-Key XCBC

Following the work by Black and Rogaway, Moriai and Imai enhanced Lemma 4 in [BR00] and introduced a new lemma [MI02b]. The enhanced lemma proves that no adversary efficiently distinguishes a pair of two functions, $(P_{K_1}(\cdot), P_{K_2}(\cdot))$ and $(P_{K_1}(K_1 \oplus \cdot), P_{K_1}(\cdot))$. Based on this lemma, the two PRPs, P_{K_2} and P_{K_3} of FCBC, can be replaced by $P_{K_1}(\cdot \oplus K_1)$ and $P_{K_1}(\cdot \oplus K_2)$. While this enhanced scheme still achieves the provable security, one key material out of three in 3-key XCBC can be saved. Of course, the number of key setup is still optimal. One of the examples using this lemma is the 2-key XCBC proposed by Moriai and Imai [MI02a,MI02b].

2-key XCBC in [MI02b]: Share two key materials, K_1 for a secret parameter for PRP and the other b-bit K_2 for secret masking. If a message M satisfies $len(M) > 0$ and $b \mid len(M)$, then set $K_t = K_2$. Otherwise, set $K_t = K_1$ and pad the minimum binary string 10^i to M such that the padded message $\tilde{M}$ fits

[1] More specifically, for a fixed length of message, only two keys are always required. Depending on the length of message, either of two keys is used as the second key material.

the b-bit block boundary, or more precisely pad the bit string $10^{b-1-len(M) \bmod b}$ to M. Let M^- be the one-block truncated $\tilde{M}$. The 2-key XCBC generates the tag_M as MAC that is calculated in the following procedures:

$$tmp = CBCMAC_{K_1}(M^-), \qquad tag_M = P_{K_1}(tmp \oplus K_t).$$

This scheme successfully reduces the required key material in comparison with 3-key XCBC. We summarize the performance of 2-key XCBC.
Security. Assuming that the security of PRP and the randomness of K_t, namely K_1 and K_2, it is proven that the 2-key XCBC cannot be forged. The security proof of the 2-key XCBC is based on the pseudo-randomness of 2-key XCBC function, which immediately implies the unforgeability.
Workload. The number of P invocations is $\lceil m/b \rceil$. This means that the 2-key XCBC is almost as efficient as CBC-MAC from the viewpoint of required computational complexity. Moreover thinking of that a PRP in a scheme is replaced to a secure block cipher in reality, consideration of the number of keys fed to key schedule is also important. In this regard, the scheme is also optimally efficient since the number of keys parameterizing PRP is one.
Key materials. The most important advantage of 2-key XCBC is the number of key materials. Only the original CBC-MAC (but with major security problems) achieves single-key scheme. The EMAC scheme uses two key materials but there are some disadvantages over 2-key XCBC: one necessary additional P invocation, no definition dealing with a message that does not fit b-bit block boundary (EMAC* does, but it requires another P invocation), and two keys are always set to PRP.

4 2-Key XCBC Variations and Raw-Key Masking

In this part, we discuss the 2-key XCBC in more detail. The original version of 2-key XCBC was introduced in the previous part of the paper. However a number of variations can be also considered. Firstly we formalize a simple variation that is implicitly described in [MI02b].
2-key XCBC′: Share two key materials, K_1 for a secret parameter for PRP and the other b-bit K_2 for secret masking. If a message M satisfies $len(M) > 0$ and $b \mid len(M)$, $K_t = K_1$. Otherwise, $K_t = K_2$ and pad the minimum binary string 10^i to M such that the padded message $\tilde{M}$ fits the b-bit block boundary, or more precisely pad the bit string $10^{b-1-len(M) \bmod b}$. Let M^- be one-block truncated $\tilde{M}$. The 2-key XCBC generates the tag_M in the same way as 2-key XCBC. Note that the difference from the original 2-key XCBC is the definition of K_t. Moriai and Imai also note that this variation also achieves the provable security [MI02b].

In addition to this variation, there are some variations for the operation to K_t. For the simplest case, we can consider the case where $len(K_1) \not= b$. If $len(K_1) > b$, K_t can be generated out of the truncate of K_1. If $len(K_1) < b$, K_t can be a string of K_1 padded with a constant or a part of K_1. Another possible variations are ones differing the replacement to xor operation, e.g., addition

and subtraction operations. Moreover any of these variations have another yet variations by defining K_t in the opposite manner, like 2-key XCBC′.
Raw-key masking. In order to discuss the security of 2-key XCBC variations, we especially focus on the security against a certain special form of plaintexts. In order to define these plaintexts, we introduce a way to categorize all 2-key XCBC variants into two groups; one is called 2-key XCBC group and the other 2-key XCBC′ group. The 2-key XCBC group includes 2-key XCBC variations where K_t is generated out of K_2 for a message M with $len(M) > 0$ and $b \mid len(M)$. The 2-key XCBC′ group is the co-group of 2-key XCBC group, i.e., that includes 2-key XCBC variations where K_t is generated out of K_1 for a message M with $len(M) > 0$ and $b \mid len(M)$.

Following these notations, we define the target plaintexts for each group. For 2-key XCBC group, let $\mathcal{M}$ to be the set of messages with $0 \leq len(M) < b$. Note that the size of $\mathcal{M}$ is sufficiently large since $\sum_{i=0}^{b-1} 2^i = 2^b - 1$. For 2-key XCBC′ group, let $\mathcal{M}$ to be the set of messages with $len(M) = b$. Also note that the size of $\mathcal{M}$ in this case is sufficiently large since it is 2^b.

The purpose of these target plaintexts is to construct a situation of *raw-key masking* where the PRP parameter K_1 is reused to mask the message. Therefore regardless of which category one is considering, the MAC generation against a message $M \in \mathcal{M}$ can be written as: $tag_M = P_{K_1}(f(M, K_1))$, where f is a simple non-cryptographic function, e.g., xor, addition and subtraction operations. The operands can be also the result of a simple operation such as $\lll$ and bitwise negation.

In the following part of the paper, we study the security of the function $P_{K_1}(f(M, K_1))$, treating some real examples of P and examples of f[2].

5 Security Analysis: 2-Key XCBC with PRP Constructions

We discuss the security of $tag_M = P_{K_1}(f(M, K_1))$. Especially we treat the cases in which P is a PRP construction with a provable security. To respect the practicality of the discussion, we deal with f functions that is naturally defined ones, such as xor or truncation in a natural manner.

As we already mentioned, we concentrate on a message $M \in \mathcal{M}$. Thanks to the flexible definition of $\mathcal{M}$, we can treat any variants suggested in the previous as a function of $tag_M = P_{K_1}(f(M, K_1))$. Since an adversary against a MAC scheme can gather sufficient pair of a message and its valid tag $(M^{(i)}, tag^{(i)})$, the adversary is equivalent to an adversary against $tag_M = P_{K_1}(f(M, K_1))$ in chosen-plaintext-attack circumstance.

[2] In our situation, an adversary does not have the oracle by which $P_{K_1}^{-1}(C)$ is computable in response to C. This oracle may not exist in the genuine implementation. However if ever a wrong implementation happens to allow to use such oracle, the scheme is immediately cracked. Such circumstances could be realized in the case when K_1 is reused for encryption (e.g, CBC mode). We do not treat this situation since the discussion is trivial and out of the scope in this paper.

5.1 Even-Mansour Construction

A publicly-known PRP can be efficiently transformed to a provably-secure secret PRP by Even-Mansour construction [EM97]. This construction is effective and increases the efficiency very much especially in the case that the secret key is frequently changed in the memory-limited device. Moreover the throughput of the data randomizing part can be also improved. Since the secret PRP can be used for many cryptographic schemes such as a MAC generation and a symmetric-key encryption, many cryptographic implementations (especially where keys are frequently changed) take advantage of EM construction using a public PRP.

The security of EM construction is proven [EM97]. The length of the secret information required for EM construction is $2b$ bits where b is the block length of the public PRP. Then the proven security of EM construction can achieve the effective key length of $b - l - m$ bits against an adversary the number of whose oracle call to the public PRP and EM construction is limited to 2^m and 2^l respectively.

The description of EM construction is as follows. There are parameters of EM construction, namely a public b-bit PRP P and two b-bit keys K_a and K_b. The EM construction can be written as

$$EM_{P,K_a,K_b}(M) = P(M \oplus K_a) \oplus K_b.$$

When EM construction is used as a block cipher, the computational complexity for key setup is almost negligible. Therefore, if a scheme where a PRP is keyed by plural key materials (for instance FCBC [BR00]) is concerned, there is an advantage in key setup efficiency over the current secure block ciphers.

We discuss the security of the 2-key XCBC mode with a secret PRP constructed by EM construction. An adversary can obtain pairs of a message and its valid tag $(M^{(i)}, tag^{(i)})$. Based on the knowledge of $(M^{(i)}, tag^{(i)})$, an adversary tries to generate a message M' that is different from any $M^{(i)}$, and its valid tag tag'.

We treat special 2-key XCBC variants in which raw-key masking material is defined by the K_a of EM construction. Especially we consider these two schemes, $2kXCBC_{EM_{P,K_a,K_b},K_a,K_3,\oplus}$ and $2kXCBC'_{EM_{P,K_a,K_b},K_3,K_a,\oplus}$. These variations are ones of most possible variants since the length of K_a fits to the requirement of raw-key masking for 2-key XCBC and 2-key XCBC′. We give the following lemma concerning the security of these schemes. The lemma treats only 2-key XCBC but the same fact is also applied to 2-key XCBC′ as well.

Lemma 1. *(Security of 2-key XCBC with Even-Mansour construction) The security of 2-key XCBC is insecure if raw-key masking material of 2-key XCBC is defined to be K_a, the plaintext-side key material of Even-Mansour construction. More specifically, there is an attacker with one MAC-generating oracle call and two public-PRP oracle call. The forgery requires only negligible computational complexity and memory.*

Proof. We show an algorithm to forge 2-key XCBC with the specified EM construction. For simplicity, we show the attack against 2-key XCBC specified in

[MI02b]. However, almost the same attack can be applied to the 2-key XCBC′ scheme with EM construction.

Attack algorithm

Precalculation

P1. An adversary against 2-key XCBC chooses any message $M^{(1)}$ such that $0 \le len(M^{(1)}) < b$ and obtain its valid tag $tag^{(1)}$.

P2. The adversary generates the padded message $\tilde{M}^{(1)} = M^{(1)} \mid\mid 10^i$ where $b \mid len(\tilde{M}^{(1)})$ and $0 \le i < b$.

P3. The adversary calculates $tag' = P(\tilde{M}^{(1)})$ since P is publicly known.

P4. Set $K'_b = tag^{(1)} \oplus tag'$.

Forgery

F1. Generate a message $M^{(2)}$ such that $0 \le len(M^{(2)}) < b$ and $M^{(2)} \ne M^{(1)}$.

F2. Generate the padded message $\tilde{M}^{(2)} = M^{(2)} \mid\mid 10^i$ where $b \mid len(\tilde{M}^{(2)})$ and $0 \le i < b$.

F3. Generate the tag for $M^{(2)}$, $tag^{(2)} = P(\tilde{M}^{(2)}) \oplus K'_b$.

Attack analysis
Since the adversary choose $M^{(1)}$ so as to hold $0 \le len(M^{(1)}) < b$, $M^{(1)}$ will be internally padded and generate the same string as $\tilde{M}^{(1)}$. From the same reason, 2-key XCBC will choose K_a for raw-key masking. Therefore the valid tag for $M^{(1)}$ is mathematically described as follows:

$$\begin{aligned} tag_{M^{(1)}} &= EM_{P,K_a,K_b}(\tilde{M}^{(1)} \oplus K_t) \\ &= P(\tilde{M}^{(1)} \oplus K_a \oplus K_a) \oplus K_b \\ &= P(\tilde{M}^{(1)}) \oplus K_b. \end{aligned}$$

Since the adversary knows $tag' = P(\tilde{M}^{(1)})$,

$$\begin{aligned} tag' \oplus tag_{M^{(1)}} &= P(\tilde{M}^{(1)}) \oplus (P(\tilde{M}^{(1)}) \oplus K_b) \\ &= K_b. \end{aligned}$$

Therefore K'_b that the adversary calculates is equal to K_b.

We now verify that $tag^{(2)}$ is a valid tag for $M^{(2)}$. Since $M^{(2)}$ is also chosen so as to hold $0 \le len(M^{(2)}) < b$, the valid tag for $M^{(2)}$ is formulated as follows:

$$\begin{aligned} tag_{M^{(2)}} &= EM_{P,K_a,K_b}(\tilde{M}^{(2)} \oplus K_t) \\ &= P(\tilde{M}^{(2)} \oplus K_a \oplus K_a) \oplus K_b \\ &= P(\tilde{M}^{(2)}) \oplus K_b. \end{aligned}$$

The tag $tag^{(2)}$ generated by the adversary coincides to $tag_{M^{(2)}}$. □

This kind of attack can be applied because of the potential weakness due to the key masking at outside of the PRP. We show the attack against the weakest variation of f but the most possible specification (if this kind of security has not been concerned). There are number of possibilities to the variation of the raw-key masking function f since the purpose of f is only to mask the data. However another elaborated attack can be also applied to some of variations.

For f functions such that $f(M, K) = M \oplus (K \lll_b c)$ for some c, the adversary is required an additional workload to exhaustively guess the value $k \oplus (k \lll_b c)$. However in this case the total workload is still equal to(when b and c are coprime to each other) or less than (otherwise) the proven security level to EM scheme.

5.2 DESX Construction

DESX is an efficient way of improving the security of a block cipher with small additional computational complexity and additional key resources. This was originally proposed by Rivest but not published. The formal security level was proven by Kilian and Rogaway [KR96].

There are two variations of DESX construction; we introduce the 3-key variant at first so that the other variant can be seen as the special case of the former. Let P_{K_a} be a b-bit PRP parameterized by the secret key K_a. Then 3-key DESX takes another two b-bit key materials, namely K_b and K_c. The scheme can be written as:

$$DESX_{P_{K_a}, K_b, K_c}(M) = P_{K_a}(K_b \oplus M) \oplus K_c. \tag{1}$$

The 2-key variant is the case in which $K_b = K_c$ holds.

As one can easily find the similarity between the structure of DESX and EM construction, we can apply the similar attack against a certain special case (but still very possible) of DESX. However, unlike the case with EM construction, our attack only violates the security bound of DESX. That is, our forgery against 2-key XCBC scheme with DESX requires partial exhaustive key search over K_a. We summarize our result by showing the similar lemma. First of all, we formalize the target construction of 2-key XCBC with DESX. Let P_{K_a} be any publicly known PRP (e.g. a block cipher) keyed by the secret key K_a. The DESX construction we study here is defined in Equation (1). Then immediately we can formalize the special case of the 2-key XCBC with DESX construction as follows:

$$2kXCBC_{DESX_{P_{K_a}, K_b, K_c}, K_b, K_3}.$$

We analyze the MAC scheme.

Lemma 2. *(Security of 2-key XCBC with DESX construction) The security of 2-key XCBC cannot be bounded by the security of DESX if plaintext-side key material of DESX (that is K_b in the previous definition of DESX) is set to be one of the key material used in 2-key XCBC. More specifically, there is an attacker that performs a forgery with almost the same time complexity to what is required to the key exhaustive search for K_a.*

Proof. The proof for this lemma is basically the same to the proof of Lemma 1 except the additional exhaustive search over K_a. We will only describe the attack algorithm and omit the detail analysis.

Attack algorithm

Precalculation

P1. An adversary against 2-key XCBC chooses any messages $M^{(i)}$ $(i = 1, \cdots 2)$ such that $0 \leq len(M^{(i)}) < b$ and obtain its valid tag $tag^{(i)}$.

P2. The adversary generates the padded message $\tilde{M}^{(1)} = M^{(1)} \mid\mid 10^i$ where $b \mid len(\tilde{M}^{(1)})$ and $0 \leq i < b$.

P3. Do the exhaustive search over possible K_a and find the correct pair of (K_a, K_c) as follows:

1. For each K_a's candidate, $K_a^{(i)}$, set $K_c^{(i)} = P_{K_a^{(i)}}(M^{(1)}) \oplus tag^{(1)}$.
2. If $P_{K_a^{(i)}}(M^{(2)}) \oplus K_c^{(i)} == tag^{(2)}$, break the loop and set $(K'_a, K'_c) = (K_a^{(i)}, K_c^{(i)})$.

Forgery

F1. Generate M' such that $0 \leq len(M') < b$ and $M' \neq M^{(i)}$ for any i.

F2. Set $\tilde{M}' = M' \mid\mid 10^i$, where $b \mid len(\tilde{M}')$ and $0 \leq i < b$.

F3. Generate the tag for M', $tag' = P_{K'_a}(\tilde{M}') \oplus K'_c$.

$\square$

Roughly speaking the use of DESX in this case cannot bring any efficient security advantage over the normal block cipher usage, P_{K_a}. The required data for this attack are a few pair of a message and its tag. Although the required time complexity for this attack is almost the same as the key exhaustive search for underlying block cipher, the amount of the required computational complexity is remarkably below than what is proven for DESX. More specifically we require a constant number of invocations to $DESX_{P_{K_a}, K_b.K_c}$, 2^k off-line calculation of P, and negligible memory space.

6 Security Analysis: 2-Key XCBC with Real Block Ciphers

In the previous section, we exemplified the special case of 2-key XCBC such that the mode of operation potentially weakens the underlying primitive. However the similar discussion can be also applied to a block-cipher primitive, whose security is typically evaluated in a heuristic manner.

One of the example is about the AES cipher [FIPS197], by which we exemplify the case where 2-key XCBC mode *weakens* the AES cipher from the view point of cryptographically effective number of rounds. The second example is on a future promising cipher Camellia [AI+00]. We also discuss the security of Camellia used in 2-key XCBC. In Camellia's case, we conclude that 2-key XCBC does not weaken the cryptographic security evaluated from the key schedule analytic approach. However, our security evaluation in this respect is not trivial because we use some of Camellia's key-schedule analyses.

6.1 AES with 2-Key XCBC

AES [FIPS197] (originally proposed as Rijndael [DR99]) is a 128-bit block cipher whose key length is 128, 192 or 256 bit. The number of rounds r is defined to be 10, 12 and 14 for the key length 128, 192 and 256 bit. The key scheduling algorithm generates $(r + 1)$ 128-bit round keys. The first round key is identical to the most significant 128 bits in the secret key.

To encrypt a plaintext block, the input is xored with the first round key followed by the non-linear substitution, shift-row, mix-column and key addition for the round one. At the last round (the tenth round for 128-bit key), there is no mixcolumn operation.

We study the 2-key XCBC with AES in more detail. For a message $M \in \mathcal{M}$, any variation of 2-key XCBC generate a tag calculated by $tag_M = P_{K_1}(f(M, K_1))$. We let the non-cryptographic function f to be $f_{high^{128}\cdot\oplus}$, the xor operation of b-bit M and the most significant 128 bits of K_1. Note that this definition of the raw-key masking K_1 is a fairly natural definition rather than ones elaborated to embed a security flaw.

In this case, the initial whitening of AES is the exactly the same operation as the raw-key masking due to the 2-key XCBC mode. Therefore these two operations cancel out each other. Consequently the padded message $\tilde{M}$ is the exact value after the key whitening. Since AES does not have key addition until the key addition at the end of the first round, the adversary can take an advantage of one round due to the secret-free recovery of intermediate value. This means that a chosen-plaintext attack against $(r - 1)$-round AES variant is sufficient to attack such 2-key XCBC mode.

The current security assessment of a block cipher is considerably related to the number of rounds. Many cryptographic techniques exploit a certain kind of characteristics in a round and try to extend the characteristics to as many rounds as possible. Therefore, from that common point of view, 2-key XCBC with AES is cryptographically weaken the underlying AES cipher, especially in a context of the current state-of-the-art cryptography.

6.2 Camellia with 2-Key XCBC

Camellia is a 128-bit block cipher, that supports 128-, 192- and 256-bit key lengths [AI+00]. Camellia adopts Feistel structure in its overall design. However there are additional structural components; initial and final whitening and FL and FL^{-1} functions. In the key-scheduling algorithm, two intermediate keys K_L and K_A (in addition, K_R and K_B for 192 and 256 key length) are generated out of the secret key. Each round key is a upper or lower half of bitwisely-rotated intermediate keys.

If Camellia is used with 2-key XCBC mode, the cryptographic security must be considered especially concentrating on the function $Camellia_{K_1}(f(M, K_1))$. As we noted, the plaintext is initially whitened by the 128-bit initial whitening key (kw_1, kw_2). Note that the specification of the key schedule defines (kw_1, kw_2) such that $kw_1 \mid\mid kw_2 = K_1$ for 128-bit key length. For other key length, the same thing can be found when we define $f = f_{high128 \cdot \oplus}$. Therefore the intermediate value after the whitening coincides to the padded message in a result of canceling out the raw-key masking and the initial whitening. Therefore in this mode, the effect of the initial masking disappears.

This characteristics can be regarded as the same thing as what happened in AES. However, the discussion in the following is different. Unlike to the case of AES, Camellia has another key masking for every input of the non-linear round function. Therefore the adversary cannot know the actual input of non-linear functions. Consequently, if the security of a block cipher is considered just from the viewpoint of the number of rounds, the fact that the whitening disappears will not weaken the security statement.

However, because of the characteristics due to Feistel structure, we have to consider the security against equivalent keys. In Feistel network, the initial and final whitening keys effect every round key in a linear manner. Therefore these whitening keys are transformed to each round function and can be combined with the round key [Kn91]. This observation is extremely critical if the security in the context of equivalent keys is considered. For example, LOKI89 [BKS89] has a structure with 16 equivalent keys for all keys [Kn91]. However, these equivalent keys will not happen for LOKI89 without the initial whitening. From the study of LOKI89, how the existence of the initial whitening strengthen (or weaken) the cryptographic security is not obvious.

In the following part of this section, we analyze the security of Camellia without the initial whitening. We treat the equivalent key or similar characteristics in the structure. In order to make the discussion simple, we treat the equivalent cipher, where the final whitening is combined with every round key. The resultant round keys are called pseudo round keys PK_i. If a distinct secret key pair (K, K') is a equivalent key, the key differential value $\Delta_K = K \oplus K'$ does not affect to the PK_i differentials, i.e., $\Delta_{PK_i} = 0$ for all i.

We study the key schedule of Camellia. For simplicity, we treat the key schedule for 128-bit key. The similar analysis can be also applied to 192- and 256-bit key schedules. The Camellia's key schedule for a 128-bit secret key generates 128-bit K_A value using non-linear transformations out of the secret key K_L. Please refer to [AI+00] for definitions of each round keys. From the definition some of important pseudo round keys are defined as follows:

$$PK_{13} = (K_L \lll_{128} 94)_L \oplus kw4, \qquad PK_{17} = (K_L \lll_{128} 111)_L \oplus kw4,$$

$$PK_{14} = (K_L \lll_{128} 94)_R \oplus kw3, \qquad PK_{18} = (K_L \lll_{128} 111)_R \oplus kw3.$$

where $kw4 = (K_A \lll_{128} 111)_R$ and $kw3 = (K_A \lll_{128} 111)_L$.

In order to find a remarkable characteristics for two distinct keys, the differential of all pseudo-keys Δ_{PK_i} must be zero. For necessary condition we show the followings; Two of the whitening differentials for a fixed pair (K, K'), Δ_{kw4} (and Δ_{kw3}), must cancel out round-key differentials for round 13 and 17 (or 14 and 18). This is formulated as:

$$\Delta_{K_{13}} = \Delta_{K_{17}} = \Delta_{kw4}, \qquad \Delta_{K_{14}} = \Delta_{K_{18}} = \Delta_{kw3}.$$

These are combined into one 128-bit relation as follows.

$$(\Delta_{K_L} \lll_{128} 94) = (\Delta_{K_L} \lll_{128} 111) = (\Delta_{K_A} \lll_{128} 111).$$

Note that for a fixed (K, K'), the intermediate-key differentials Δ_{K_L} and Δ_{K_A} are also fixed. Since the rotation offsets between two K_Ls differ in 17 bit (that is coprime to the register size 128), the only satisfiable differentials for K_L and K_A are only the bit string of 1^{128} for both differentials.

Since K_A is generated out of K_L by means of four-round iterative function used in the data randomizing part. With respect to the classic differential propagation, the differential probability for the corresponding Δ_{K_L} and Δ_{K_A} must be extremely small. This means that almost negligible pair of secret key must collide the key differentials at round 13, 14, 17 and 18 rounds. From this fact, the Camellia without the initial key whitening will not have remarkably large class of keys that have equivalent keys.

7 Conclusion

We studied the security of 2-key XCBC mode and demonstrated two important cases where a PRP is replaced to a provably secure PRP construction. For Even-Mansour construction, there is an efficient forgery, and for DESX construction, its probable security is compromised. The 2-key XCBC MAC generation scheme is not always secure even with a provably secure primitive.

These security-proof violation comes from abuse of raw-key masking outside of the cipher. In both cases we showed that an adversary can make the randomizing function contain two raw-key masking. In worst cases, these two raw-key maskings may cancel out each other effects.

The discussion on the security of a block cipher with raw-key masking becomes more complicated. As shown in Camellia case, a certain kind of attacks must be revisited. So to use raw-key masking safely there must be good evaluation on its primitive and scheme.

We remark that our observations discussed in this paper are not applicable to the most recent works by Kurosawa and Iwata [KI02].

References

AI+00. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, T. Tokita, "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms —Design and Analysis," In Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Proceedings, LNCS 2012, Springer-Verlag, 2001.

BD+97. M. Bellare, A. Desai, E. Jokipii, P. Rogaway, "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation," *Proceedings of the 38th Symposium on Foundations of Computer Science, IEEE*, 1997.

BKR94. M. Bellare, J. Kilian, P. Rogaway, "The Security of Cipher Block Chaining," *Advances in Cryptology, -CRYPTO'94, LNCS Vol. 839, Springer-Verlag*, 1994.

BKS89. L. Brown, J. Pieprzyk, J. Seberry, "LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications," *Advances in Cryptology - AUSCRYPT '90*, Springer-Verlag, Lecture Notes in Computer Science Vol. 453, 1990.

BR00. J. Black, P. Rogaway, "CBC MACs for arbitrary-length messages: The three-key constructions," *Advances in Cryptology, -CRYPTO2000, LNCS 1880, Springer-Verlag*, 2000.

BR02. J. Black, P. Rogaway, "A Block-Cipher Mode of Operation for Parallelizable Message Authentication," *Advances in Cryptology, -EUROCRYPT 2002, LNCS Vol. 2332, Springer- Verlag*, 2002.

BW00. A. Biryukov, D. Wagner, "Advanced Slide attacks," *Advances in Cryptology, -EUROCRYPT 2000, LNCS Vol. 1807, Springer-Verlag*, 2000.

Da93. J. Daemen, "Limitations of the Even-Mansour construction," *Advances in Cryptology - ASIACRYPT'91, LNCS, Vol. 739,* Springer-Verlag, 1993.

DR99. J. Daemen, V. Rijmen, *AES Proposal: Rijndael*, AES Algorithm Submission, September 3, 1999, available at `http://www.nist.gov/CryptoToolkit.`

EM97. S. Even, Y. Mansour, "A construction of a cipher from a single pseudorandom permutation," *J of Cryptology, 10(3) 151-161*, Summer 1997.

FIPS46-3. National Institute of Standards and Technology, Federal Information Processing Standards Publication 46-3, Data Encryption Standard (DES).

FIPS81. National Institute of Standards and Technology, Federal Information Processing Standards Publication 81, DES Modes of Operation (DES), 1980.

FIPS197. National Institute of Standards and Technology, Federal Information Processing Standards Publication 197, Advanced Encryption Standard (AES).

GGM86. O. Goldreich and S. Goldwasser and S. Micali, "How to Construct Random Functions," *Journal of the ACM*, 33(4), 1986, 792-807.

JJV02. E. Jaulmes, A. Joux, F. Valette, "On the security of randomized CBC-MAC beyond the birthday paradox limit: a new construction," in the Preproceedings of the Fast Software Encryption 2002, Leuven, Belgium, 2002.

Kn91. L.R. Knudsen, "Cryptanalysis of LOKI," *Advances in Cryptology– ASIACRYPT '91*, Springer-Verlag, 1993, pp. 22–35.

KR96. J. Kilian, P. Rogaway, "How to protect DES against exhaustive search (an analysis of DESX)," *Advances in Cryptology - CRYPTO'96, Lecture Notes in Computer Science, Vol. 1190,* Springer-Verlag, 1996.

KI02. K. Kurosawa, T. Iwata, "TMAC, Two-Key CBC MAC," ePrint archive: Report 2002/092, available at `http://eprint.iacr.org/2002/092/`.

LR88. M. Luby, C. Rackoff, "How to Construct Pseudorandom Permutations from Pseudorandom Functions," *SIAM J. Comput.*, Vol. 17, No. 2, April 1988.

Mi02. C.J. Mitchell, "The security of two-key DESX," COSIC Seminar, Katholieke Universiteit Leuven, 15th March 2002, Leuven, Belgium.

MI02a. S. Moriai, H. Imai, "2-Key XCBC: The CBC-MAC for Arbitrary Length Messages by the Two-key Construction," a talk at the *Recent Results* session of *Fast Software Encryption 2002*, Leuven, 2002.

MI02b. S. Moriai, H. Imai, "2-Key XCBC: The CBC MAC for Arbitrary-Length Messages by the Two-Key Construction," In the Proc. of SCIS2002, The 2002 Symposium on Cryptography and Information Security, The Institute of Electronics, Information and Communication Engineers, 2002 (in Japanese).

RACE. A. Berendschot, B. den Boer, J. Boly, A. Bosselaers, J. Brandt, D. Chaum, I. Damgaard, M. Dichtl, W. Fumy, M. van der Ham, C. Jansen, P. Landrock, B. Preneel, G. Roelofsen, P. de Rooij, J. Vandewalle, *Final Report of Race Integrity Primitives, Lecture Notes in Computer Science, Vol. 1007*, Springer-Verlag, 1995.

A The Important Lemma Constructing 2-Key XCBC

Settings. Let us think of the situation where an adversary tries to distinguish whether the oracle outputs based on the pair of functions $(\pi_1(\cdot), \pi_2(\cdot))$ or the other pair of functions $(\pi(\cdot), \pi(K \oplus \cdot))$ in response to the adversary's queries.

Lemma 3. *(Lemma 1 from [MI02b]) Let n be a positive integer and define $N = 2^n$. If an adversary makes at most p queries to the oracle, we have the following relation concerning the advantage of the adversary.*

$$\Big| \Pr[\pi_K \overset{R}{\leftarrow} \mathrm{Perm}(n) : \mathcal{A}^{\pi_K(\cdot), \pi_K(K \oplus \cdot)} = 1] - \Pr[\pi_1, \pi_2 \overset{R}{\leftarrow} \mathrm{Perm}(n) : \mathcal{A}^{\pi_1(\cdot), \pi_2(\cdot)} = 1] \Big| \le \frac{p^2}{N}.$$

A New Statistical Testing for Symmetric Ciphers and Hash Functions

Eric Filiol

INRIA, Projet CODES
B.P. 105, 78153 Le Chesnay Cédex, France
Eric.Filiol@inria.fr

Abstract. This paper presents a new statistical testing of symmetric ciphers and hash functions which allow us to detect biases in a few of these systems. We first give a complete characterization of the Algebraic Normal Form (ANF) of random Boolean functions by means of the Möbius transform. Output bits of a cryptosystem are here described by a set of Boolean functions. The new testing is based on the comparison between their Algebraic Normal Form and those of purely random Boolean functions. Detailed testing results on several cryptosystems are presented. As a main result we show that AES, DES, Snow, and Lili-128 fail the tests wholly or partly and thus present strong biases.

Keywords: Boolean function, statistical testing, symmetric cipher, randomness, hash function, Möbius transform, Walsh Transform.

1 Introduction

Randomness is the ground property of cryptography. For the attacker, any quantities produced by a given cryptosystem must look as unpredictable as possible. It means that these quantities have to be of sufficient size and "be random" in the sense that the probability of any particular value being selected must be as weak as possible to preclude a cryptanalyst from gaining advantage through optimed search strategy based on such probability [15, p 169].

From a general point of view, any symmetric cipher and any hash function must be designed as a pseudorandom bit generator (PRBG) relatively to each of its output bits.

Two important requirements are then to be satisfied: the output sequences of a PRBG must be statistically indistinguishable from truly random sequences and the output bits must be unpredictable to an attacker with limited computing facilities. Therefore, many different statistical tests have been proposed and are usually implemented to evaluate these two requirements. Historically, we must cite Golomb's randomness postulates [11]. These tests have been designed as necessary but not sufficient tests to check if a shift register sequence statistically behaves properly. Yet statistically good according to these postulates, this kind of sequence has been shown very predictable when using the Berlekamp-Massey algorithm [16]. This is the illustration that randomness is uniquely defined relatively to the statistical tests we may use.

Many other statistical tests have been proposed in order to improve what may be considered as “random”. Among many others, let us cite those that are mainly implemented: frequency test, serial test, poker test, runs test and autocorrelation test [8,13], Maurer’s universal statistical test [17], (for a more detailed bibliography on statistical tests used in cryptography see [15, pp 188-189]).

All the recently proposed symmetric cryptosystems and hash functions can be considered as satisfying all the known randomness requirements. Now the essential part of the cryptanalyst’s work is to find an exploitable bias, due to an unknown design flaw, that none of the up-to-now known test detected. For that, the cryptanalyst generally first designs a new hypothesis testing based on a new test. Let us recall that in fact randomness is a theoretical indeed “philosophical” concept. Practically speaking, it can only be determined and defined relatively to the set of statistical tests used to evaluate it.

In this paper we present a new hypothesis testing based on a χ^2 distribution and called Statistical Möbius Analysis. More precisely, we define as working statistic X the number of monomials of degree exactly d in the *Algebraic Normal Form* (ANF) of all the Boolean functions modeling each of the output bits. The set of these d-monomials which are effectively represented in the ANF, are practically computed by means of the Möbius transform. A secure cryptosystem has a fixed distribution determined by general results on random Boolean functions. Then one-sided tests allow us to check if the constituent Boolean functions are truly random.

These tests have been implemented for a few recently proposed stream ciphers and block ciphers, as well as for the main hash functions. All are known to have passed the previously known statistical tests and thus are considered as having very good random properties. Our main results is that famous cryptosystems such AES, DES, Snow and Lili-128 did not pass our tests, wholly or partly. Other results as well as detailed data will be found in [5].

This paper is organized as follows. Section 2 presents the necessary preliminaries and gives the characterization of the Algebraic Normal Form (ANF) of random Boolean functions. In particular, we complete the results presented in [19], make them more practical and give new results on the total degree of a Boolean function. Section 3 presents the new test we designed whilst Section 4 gives detailed numerical results that have been obtained for a few stream ciphers (Lili-128, Snow, BGML and RC4), block ciphers (DES and AES) and hash functions (SHA-0, SHA-1, Ripe-MD, Ripemd160, Haval, MD4 and MD5).

2 Characterization of Boolean Functions and Results

In this section, we present a new statistical way of describing a Boolean function by use of its ANF. This latter can be uniquely computed by means of the Möbius transform. We deduce results on the balancedness and correlation properties with the help of the Walsh transform.

2.1 Structure of the Algebraic Normal Form

A Boolean function is a function f from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The number of such functions is 2^{2^n}. We define a random Boolean function as a function f whose values are independent, identically distributed (*i.i.d.*) random variables that is to say

$$\forall(x_1,\ldots,x_n)\in\mathbb{F}_2^n,\quad P[f(x_1,\ldots,x_n)=0]=\frac{1}{2}. \tag{1}$$

In other words, every $f(x_1,\ldots,x_n)$ is a *Bernoulli* random variable of parameter $\frac{1}{2}$. The corresponding probabilistic law will be denoted $\mathcal{B}(p)$ whith $p=\frac{1}{2}$ in our present case[1].

The weight of a Boolean function over $\mathbb{F}_2^n$ is defined by $wt(f) = |\{x\in\mathbb{F}_2^n | f(x)=1\}|$. Then a Boolean function will be said to be *balanced* if $wt(f)=2^{n-1}$. Note that a random Boolean function, as defined above, may be not balanced. In fact we will give the general probability for such a function to be balanced.

The *Algebraic Normal Form* (ANF) of f is the multivariate polynomial given by $f(x_1,\ldots,x_n)=\bigoplus_{u\in\mathbb{F}_2^n} a_u x^u,\quad a_u\in\mathbb{F}_2$, where $u=(u_1,\ldots,u_n)$ and $x^u=\prod_{i=1}^n x_i^{u_i}$. The a_u are given by the Möbius transform [14] of f:

$$a_u=\bigoplus_{x\preceq u} f(x) \tag{2}$$

where $\preceq$ denotes the partial order on the Boolean lattice, that is to say that $\alpha\preceq\beta$ if and only if $\alpha_i\le\beta_i$ for all $1\le i\le n$. A monomial $a_u x^u$ of the ANF will then be said of degree k if $a_u=1$ and if $wt(u)=k$ where $wt(.)$ denotes the Hamming weight. With these notations we now can state:

Proposition 1 *The Algebraic Normal Form (ANF) of a random Boolean function f from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ has 2^{n-1} monomials in average. For every k such that $0\le k\le n$, there are an average of $\frac{1}{2}\binom{n}{k}$ monomials of degree k.*

When $k=0$ (resp. $k=n$), it is equivalent to assert that half of randomly chosen Boolean functions contains a_0 (resp $a_{(111\cdots 11)}$) in their ANF.

Proof. A given monomial $x_{i_1}x_{i_2}\ldots x_{i_k}$ of degree k will be part of the ANF if and only if $a_u=1$ where the support of u (that is to say the set of indices j such that $u_j=1$ and denoted $supp(u)$) is $\{i_1,i_2,\ldots,i_k\}$. Now we have

$$a_u=f(\overline{0}))\oplus\bigoplus_{j=1}^{k} f(e_{i_j})\oplus\left(\bigoplus_{l=1}^{k}\bigoplus_{j=1,j\neq l}^{k} f(e_{i_j}\oplus e_{i_l})\right)\oplus\ldots\oplus f(\bigoplus_{j=1}^{k} e_{i_j}), \tag{3}$$

where $\overline{0}=(0,0,\ldots,0)$ and e_i is the n-uple whose only its i-th coordinate is non zero. The right side of Equation (3) has $\sum_{j=1}^{k}\binom{k}{j}=2^k$ terms. We have

[1] Every n-tuple $(x_1,\ldots,x_n)$ is randomly and independently chosen, then $f(x_1,\ldots,x_n)$ too. It is equivalent to randomly choose f from the set of Boolean functions.

$a_u = 1$ if an odd number of terms are all equal to 1. There are 2^{k-1} such odd configurations. Each of them, according to (1) has probability $\frac{1}{2^k}$ to be equal to 1 since we consider *i.i.d.* variables. Whence we have $P[a_u = 1] = 2^{k-1} \times \frac{1}{2^k} = \frac{1}{2}$. Thus the number of monomials of degree k in the ANF will be $P[a_u = 1] \times \binom{n}{k} = \frac{1}{2} \times \binom{n}{k}$. □

We can in fact generalize this results with the following theorem:

Theorem 1 *With the notation of Proposition 1, the number n_k of monomials of degree k has normal distribution with mean value $E[n_k] = \frac{1}{2}\binom{n}{k}$ and variance $V[n_k] = \frac{1}{4}\binom{n}{k}$.*

To be mathematically rigorous, we should consider the binomial distribution instead of the normal distribution. Moreover, we should write "X tends toward normal distribution" rather than "X has normal distribution". However, probability theory [4] entitle us such shortcuts as soon as the conditions of application for the Central Limit Theorem are fulfilled. It is the case in our work.

Proof. The proof is straightforward when considering that a_u, for all $u \in \mathbb{F}_2^n$ is a Bernouilli random variable with parameter $\frac{1}{2}$, where $E[a_u] = \frac{1}{2}$ and $V[a_u] = \frac{1}{4}$. Since $n_k = \sum_{wt(u)=k} a_u$, for large enough values of the number of u of weight k, the Central Limit Theorem gives the result (as soon as $n_k \geq 30$ [4]). □

This proposition allows to study the randomness properties of a Boolean function. Let us consider a function f used for the feedback of a shift register of length L. If f is constant (its ANF has only one monomial), the output will not be random at all. In the case of the linear feedback (the ANF of f is of degree 1 and has at most n monomials), the randomness properties are limited: the linearity properties are not suppressed, and combinatorial information is easy to get (for details see [11]). Moreover, it is very easy to reconstruct the feedback polynomial with only $2L$ output bits [16]. This is due to the fact that linear functions have very limited randomness properties.

In other words, if we consider $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ such that (*e.g.*) $f(x) = f(y) = 1$, the less random the function is, the easier is the extraction of information on x and y.

Example 1 *Let us take $f(x_1, x_2) = x_1 \oplus x_2$. Any $x = (x_1, x_2)$ and $y = (y_1, y_2)$ with $x \neq y$ such that $f(x) = f(y) = 1$ will satisfy $x_1 \oplus y_1 = 1$. This comes from the fact that the values of the truth table are "structured" and not "randomly spread" into this table.*

Proposition 1 gives us the following criterion:

Corollary 1 *A Boolean function used for cryptographic applications and presenting the best trade-off in terms of its cryptographic properties must have a degree as high as possible.*

Proof. This directly comes from the fact that a n-variable random Boolean function in average has its term of degree n with probability $\frac{1}{2}$ and will contain $\frac{n}{2}$

terms of degree $n-1$. According to the upper bound of the degree [23] of a function presenting the best trade-off in terms of correlation immunity, balancedness, ..., we have for a t-correlation immune function: $\deg(f(x_1,\ldots,x_n)) \leq n-t-1$. Constraining the function with given properties lowers the algebraic degree. Combinatorial structures are introduced while randomness is lessened. In the search for the best possible trade-off, to keep good randomness properties by forbidding to get combinatorial information on the function inputs, the function should have the highest possible degree. □

2.2 Characterization of the Walsh Coefficients

The *Walsh Hadamard transform* of a Boolean function f refers to the following transformation: $\forall u \in \mathbb{F}_2^n, \quad \widehat{\chi_f}(u) = \sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)+<x,u>}$, where $<x,u>$ denotes the usual scalar product computed over $\mathbb{F}_2^n$. A well-known result allows to characterize the correlation immunity of f with the Walsh Hadamard transform:

Proposition 2 *[24] A Boolean function f is t-order correlation immune if and only if* $\forall u \in \mathbb{F}_2^n, \quad 1 \leq wt(u) \leq t \qquad \widehat{\chi_f}(u) = 0.$

Moreover f is balanced if and only if $\widehat{\chi_f}(0,0,\ldots,0) = 0$.

Proposition 3 *Let f be a random Boolean function over $\mathbb{F}_2^n$ with $n \geq 5$. For all $u \in \mathbb{F}_2^n$, $\widehat{\chi_f}(u)$ is a random variable which has Gaussian distribution with mean value* 0 *and variance* 2^n.

Proof. First we can write $\widehat{\chi_f}(u) = \sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)+<x,u>} = (2^n-2)\cdot\sum_{x\in\mathbb{F}_2^n}(f(x) + <x,u>)$. Since x and $f(x)$ are independent, we can consider $<x,u>+f(x)$ as independent, identically distributed random variables for all x as well. Let us note $Y = \sum_{x\in\mathbb{F}_2^n}(f(x)+<x,u>)$. For $n > 5$ (that is to say $2^n > 30$), due to the central limit theorem [4], Y has a Gaussian distribution $\mathcal{LG}(E,\sigma^2)$ with

$$E[Y] = 2^n P[f(x)+<x,u>=1] = 2^{n-1}$$
$$(\sigma_Y)^2 = 2^n P[f(x)+<x,u>=1]P[f(x)+<x,u>\neq 1] = 2^{n-2}.$$

Hence $\widehat{\chi_f}(u)$ has Gaussian distribution with mean value $E[\widehat{\chi_f}(u)] = 2^n(1 - 2P[f(x)+<x,u>=1]) = 0$ and variance $\sigma^2 = 4.2^n P[f(x)+<x,u>=1]P[f(x)+<x,u>\neq 1] = 2^n$. □

If Φ denotes the normal distribution function, $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}\exp\left(-\frac{t^2}{2}\right)dt$ and if $p_0 = \Phi(\frac{1}{2^{\frac{n}{2}-1}}) - \frac{1}{2}$, we then can state

Lemma 1

$$P[f \text{ balanced }] = p_0.$$

Proof. For a balanced Boolean function, we have $\widehat{\chi_f}(0,\ldots,0) = 0$. By definition, $\widehat{\chi_f}(u)$, $\forall u \in \mathbb{F}_2^n$ is even. Then we have $P[\widehat{\chi_f}(u) = 0] = P[0 < \widehat{\chi_f}(u) < 2]$. The rest is straightforward to proove with Proposition 3. □

Remark.- This result is an accurate approximation of the "exact" probability for a function to be balanced given by $p = \frac{\binom{2^n}{2^{n-1}}}{2^{2^n}}$. Table 1 compares exact probability with that computed with Theorem 1 for $5 \leq n \leq 19$. Note that computing exact probability p is highly time consuming while computation time is negligible for p_0.

Table 1. Exact and approximate probabilities for a function to be balanced

n	p	p_0	n	p	p_0	n	p	p_0
5	0.1399	0.1381	10	0.02493	0.02491	15	0.004408	0.004407
6	0.09935	0.09870	11	0.01763	0.01762	16	0.003117	0.003116
7	0.07039	0.07015	12	0.01247	0.01246	17	0.002204	0.002203
8	0.04982	0.49738	13	0.008815	0.008814	18	0.001558	0.001558
9	0.03524	0.03521	14	0.006233	0.006233	19	0.001102	0.001101

3 The New Statistical Testing

We now present the different tests we built up to evaluate new statistical properties of symmetric cryptosystems and hash functions. Let us now consider such a cryptosystem and specify the context we choose. Let there be a secret key $K = (k_0, \ldots, k_{n-1})$. A stream cipher can be seen as follows: every output bits i generated from the secret key K can be expressed by a unique ANF (by means of the Möbius transform defined by Equation (2)).

In other words, the N-bits output sequence can be described by a family of N Boolean functions $(f_t(K))_{0 \leq t < N} = (f_0(K), \ldots, f_{N-1}(K))$ where $f_i(K)$ denotes the i-th bit produced by the system and modelled as a polynomial in variables k_i (ANF). Each output bit is a Boolean function $f_t : \mathbb{F}_2^n \mapsto \mathbb{F}_2$.

Similarly, let us represent a block cipher with n-bit key K working on m-bit blocks. In the same way, but with the different output functions being evaluated on the key space and the plaintext space $P = (p_0, \ldots, p_{m-1})$, for a block cipher C, we then have $C = (c_0, \ldots, c_{m-1}) = (f_0(K, P), \ldots, f_{m-1}(K, P))$. Each of the m ciphertext bits is a Boolean function $f_t : \mathbb{F}_2^{n+m} \mapsto \mathbb{F}_2$.

A hash function $H : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ will have its m-bit message digest of block $B = (b_0, \ldots, b_{n-1})$ represented by $(h_t(B))_{0 \leq t < m} = (h_0(B), \ldots, h_{m-1}(B))$. In the rest of this paper we will use indifferently the term *output bits* and *output Boolean Functions* (or output ANFs for short) to describe the quantities produced by the cryptosystem we consider. At last we will consider that the different output Boolean functions (or bits) are statistically independant. It is precisely the result stated by previous usual, known tests.

The complete output ANF cannot be computed since it contains in average 2^{n-1} monomials. It would require exponential memory and computing time complexity. For our tests we only focus on the monomials of degree at most 3 and need only to compute the 3-truncated ANF, that is to say the partial ANF whose coefficients are effectively computed up to degree 3. For a few cases, 5-truncated ANFs have been computed when necessary. From a practical point of view, we use Formula (3) to produce them. As a result, we observe in every ANF, $\hat{n}_d$ monomials of degree exactly d.

Let us now note H_0^d the statistical hypothesis that the number $\hat{n}_d$ of monomials of degree exactly d is distributed according to the Theorem 1. In other words, the cryptosystem passes our tests and thus exhibits no particuliar structural, statistical bias for the aspect we consider when satisfying this hypothesis.

We suppose the reader is familiar with basic probability and statistics theories (for a detailed presentation see [4] and [15, Chap 5.4]).

3.1 The Affine Constant Test

Our hypothesis is then denoted H_0^0. According to Theorem 1, the probability for the affine constant a_0 to be represented in each of the output ANFs is $p = \frac{1}{2}$. Equivalently, it means that the number of output Boolean functions having $a_0 = 1$ in their ANF has normal distribution $\mathcal{N}(\frac{N}{2}, \frac{\sqrt{N}}{2})$ where N is the total number of output ANFs.

If X_S, the number of times $a_0 = 1$, is the statistic we consider over the sample output S of N ANFs, we can now describe the following two-sided test, called the *Affine Constant Test*:

1. Compute X_S over S.
2. Let us fix a significance level α (*i.e.* probability of rejecting H_0^0 when it is true) and choose a threshold x_α so that for a statistic X of normal standard distribution we have $P[X > x_\alpha] = P[X < x_\alpha] = \frac{\alpha}{2}$.
3. If the value $\hat{X}_S = \frac{X_S - \frac{N}{2}}{\frac{\sqrt{N}}{2}} > x_\alpha$ or if $\hat{X}_S < -x_\alpha$ then H_0^0 is rejected (the system fails the test) otherwise H_0^0 is kept (the system passes the test).

3.2 The d-Monomial Tests

We are now considering the monomials of degree exactly d in the output ANFs. Our testing is now denoted H_0^d.

With the notation of Theorem 1, the number of monomials of degree d in a Random Boolean Function ANF is a random variable which is $\mathcal{N}(\frac{1}{2}\binom{n}{d}, \frac{1}{2}\sqrt{\binom{n}{d}})$ distributed. We now consider two *goodness-of-fit*, one-sided tests between the expected frequencies (denoted n_d) and those (denoted $\hat{n}_d$) we observe for the considered cryptosystem.

The first test, T_1^d consider every different ANF and thus has a rather local scope by giving more weight to very weak output ANFs. The second one, T_2^d, groups the N output ANFs according to a few numbers of sets or classes. So to summarize, we will use the χ^2 distribution with ν degrees of freedom by considering the sum of the ν squared, independent random variables $\frac{(n_d^i - \hat{n}_d^i)}{\sqrt{n_d^i}}$ $(i \leq \nu)$ which have by definition standard normal distribution.

In T_1^d we have $\nu = N - 1$ (*i.e.* the number of output ANFs) while for T_2^d we choose $2 \leq \nu \leq 9$

1. Compute for each of the ν random variables n_d^i and $\hat{n}_d^i$ (n_d^i is given by applying Theorem 1).
2. Let us fix a significance level α and a threshold value x_α (computed directly from the cumulative density function of the χ^2 distribution) so that for a statistic X over a random sample we would have $P[X > x_\alpha] = \alpha$ (when X follows a χ^2 distribution with ν degrees of freedom).
3. Compute the statistics D^2 given by $D^2 = \sum_{i=1}^{\nu} \frac{(n_d^i - \hat{n}_d^i)^2}{n_d^i}$.
4. If $D^2 > x_\alpha$ then we must reject H_0^d (the system fails the test and thus presents a statistical bias) otherwise we keep H_0^d (the system does not present any significative bias).

Test T_2^d is intended to describe the considered cryptosystem from a global point of view. In particular it aims at verifying if local biases (detected with T_1^d) are still really significative at a more global level. Instead of dealing with the observed frequencies $\hat{n}_d^i$ of d-monomials for each of the N output ANFs we rather are interested with the number of output ANFs whose number $\hat{n}_d$ belongs to a given, predefined intervall $[a, b[$. The expected frequency for every class is computed from Theorem 1 by applying basic probability results.

3.3 The *d*-Monomial Tests on a Given Output ANFs Subset

Essentially, we consider the tests of Section 3.1 and 3.2 but on particuliar subsets S of output ANFs. These test are intended to detect subsets of weak output ANFs. They are denoted $T_i^d|S$ where $i = 1, 2$. Accordingly to the probability and statistics theories, results for which a given cryptosystem exhibits weaknesses must be thoroughly examined and inspected. Complementary results on sampling theory must be taken into account to discriminate "normal but extremal results" (that is to say samples S for which $P[X > x_\alpha] = \alpha$ whilst having truly random distribution) from "truly non-random behaviour".

For all these tests and in all our experiments , we considered $\alpha = 0.05, 0.01$ and 0.001.

4 Simulation Results

4.1 Stream Ciphers

We will here mainly focus on two stream ciphers that have been proposed for the NESSIE Open Call for Cryptographic Primitives [18]: Lili-128 and Snow. Other stream ciphers have been tested or are currently under testing. Table 2 summarizes results for a few of them. We considered the first $N = 6016$ output bits in our experiments.

It is worth noticing that:

– All the tested stream ciphers pass the Affine Constant test except Lili-128.
– Lili-128 exhibits extremely strong biases. Table 3 presents the results for this stream cipher. These biases have been analyzed and exploited for an operational cryptanalysis in [6].

Table 2. Stream Ciphers: Tests Results (significance levels $\alpha = 0.001$)

	T_1^1	T_1^2	T_2^1	T_2^2		T_1^1	T_1^2	T_2^1	T_2^2
Lili-128	**fail**	**fail**	**fail**	**fail**	RC4 [20]	pass	pass	pass	pass
Snow	pass	pass	**fail**	**fail**	Bgml [18]	pass	pass	pass	pass

Table 3. Lili128: Experimental results for tests T_1^d and T_2^d

	T_1^1	T_1^2	T_2^1	T_2^2
D^2	39,344.03	400,839.93	667729.02	1,028,048.45
$\chi^2_{0.001}$	6349.15			

– Snow exhibits strong biases too but only when considering global statistical behavior. Unfortunately these biases allowed us to design a complete, operationnal cryptanalysis of Snow [6].
– We can give the following interesting observations based on the comparison of the tests convergence (that is to say the distance between the estimator and the threshold value; for details see [12]). The ciphers of Table 2 can be ranked according to their relative "random" quality. We observe that ($\succeq$ means "better than") Bgml $\succeq RC4 \succeq$ Snow $\succeq\succeq$ Lili-128.
– Note that the existence of "weak keys" in stream ciphers like Lili-128 (for example all zero secret key) can only very partly explain these bad statistical results (it only affects the Affine Constant test). Snow presents bad results too whilst it does not have any weak key.
– Second version of Snow and Lili-128 exhibit the same weaknesses.

4.2 Block Ciphers

We mainly focus on the DES [7] and the AES [1]. Results for other block ciphers will be found on [5]. For block ciphers we considered both the encryption ANFs and the decryption ANFs. Since every output ANF involves both plaintext and key variables, tests T_2^d ($d = 1, 2$) have been replaced by tests T_1^d relatively to:

– the number n_1 of plaintext variables from one side and of key variables from the other side (denoted respectively $T_1^1|p$ and $T_1^1|k$).
– the number n_2 of 2-monomials respectively involving plaintext/plaintext variables, key/key variables and plaintext/key variables (tests denoted respectively $T_1^1|pp$, $T_1^1|kk$ and $T_1^1|pk$).

The DES. Table 4 gives detailed experimental results of the estimator D^2 with 63 degrees of freedom. The critical values are $\chi^2 = 82.52$ ($\alpha = 0.05$), $\chi^2 = 92.01$ ($\alpha = 0.01$) and $\chi^2 = 103.44$ ($\alpha = 0.001$).

It is worth noticing that:

Table 4. DES: Values of Estimator D^2

	T_1^1	T_1^2	$T_1^1\|p$	$T_1^1\|k$	$T_1^1\|pp$	$T_1^1\|kk$	$T_1^1\|pk$
Encr. + IP	35.06	37.65	34.75	35.57	35.41	33.47	33.25
Decr. + IP	33.68	33.93	34.75	39.74	35.41	39.12	29.95

Table 5. AES (128, 128): Values of Estimator D^2

	T_1^1	T_1^2	$T_1^1\|p$	$T_1^1\|k$	$T_1^1\|pp$	$T_1^1\|kk$	$T_1^1\|pk$
Encryption	59.61	71.32	57.84	61.51	64.47	72.34	62.39
Decryption	67.38	62.27	67.21	70.70	71.26	60.11	47.27

- DES passes the Affine Constant Test in all modes and all significance levels.
- The overall statistical quality is slightly different for encryption and for decryption (in particular the statitical results are slightly better for encryption when only the key is considered).
- DES fails the tests $T_1^1|S$ for many subsets S. For example, several 3-uples including output ANFs 0 and 22 do not pass the test. The overall results present a significant difference for the DES with or without IP. According to the results for the tests $T_1^1|S$ and $T_1^2|S$, the different modes of DES can be ranked in the following manner (($\succeq$ means "better than"):
  {DES Encr. - IP, DES Decr. + IP, DES Decr. - IP} $\succeq$ DES Encr. + IP.
  For these tests, the initial permutation IP improves the overall statistical quality for encryption only. Nevertheless IP is usually discarded by cryptology community when considering its cryptanalysis.

The AES. We will focus on the algorithm working on 128-bit blocks and with 128-bit secret key. Table 5 gives detailed experimental results of the estimator D^2 with 127 degrees of freedom. The critical values for $\alpha = 0.05$ is $\chi^2 = 159.59$ It is worth noticing that:

- AES passes the Affine Constant Test in all modes and all significance levels.
- Overall statistical quality of AES (128, 128) is good. Partial results on tests T_1^5 and T_2^5 indicate that AES do not pass the test. Moreover AES (encryption and decryption) do not pass the tests $T_1^1|S$ and $T_1^2|S$ for many subsets S. As an example, 3-uples containing output ANFs 52 and 110 are weak subsets for encryption. These biases are currently exploited to greatly improve the cryptanalysis of AES.
- Encryption and decryption exhibits quite the same overall statistical properties.

4.3 Hash Functions

We tested the following hash functions: SHA-0 [9], SHA-1 [10], Ripemd160 [3], MD4 [21], MD5 [22], Ripe-MD [2] and Haval [25] (for this latter we tested all the different versions). Extensively detailed numerical results (due to lack of space) are only available in [5]. Tests $T_1^1|S$ and $T_1^2|S$ are under way.

Table 6. Experimental results for tests T_1^d and T_2^d ($d = 1, 2$, $\alpha = 0.05$)

Hash Functions	T_1^1		T_1^2		T_2^1		T_2^2	
	D^2	χ^2	D^2	χ^2	D^2	χ^2	D^2	χ^2
SHA-1	76.87	189.52	70.89	189.52	0.04	5.99	0.42	5.99
(5,160)-haval	76.34		79.76		0.17		2.02	
Ripemd160	77.51		66.72		5.24		2.66	
(4,160)-haval	83.52		74.18		1.77		3.51	
(3,160)-haval	83.79		64.28		1.05		5.50	
SHA-0	97.08		74.50		3.26		0.42	

All the tested hash functions have passed the tests whatever may be the significance level. However we can once again give the following interesting observations based on the comparison of the tests convergence.

- The different hash functions can be ranked according to their relative "random" quality. For example when considering results of test T_1^1 (1-monomials), which is the most interesting, we have the following ordering ($\succeq$ means "better than"):
  - 160-bit Message Digest: SHA-1 $\succeq$ (5, 160)-haval $\succeq$ Ripemd160 $\succeq$ (4, 160)-haval $\succeq$ (3, 160)-haval $\succeq$ SHA-0.
  - 128-bit Message Digest: (5,128)-haval $\succeq$ Ripe-MD $\succeq$ MD5 $\succeq$ (4,128)-haval $\succeq$ (3,128)-haval $\succeq$ MD4.
- SHA-1 has indeed better statistical properties than SHA-0, especially when considering the degree 1. The inclusion of the 1-bit rotation in the block expansion from 16 to 80 words really improved the randomness properties of the hash function.
- For the Haval family, the random quality increases with the number of rounds.

Table 6 presents the results of the tests T_1^d and T_2^d for $d = 1, 2$ and for the 160-bit message digest hash functions (significance level $\alpha = 0.05$; let us recall that passing the tests for significance level α imply passing the test for $\alpha' < \alpha$ since $\chi^2_{\alpha'} > \chi^2_{\alpha}$).

5 Conclusion

This paper presents a new statistical testing of symmetric ciphers and hash functions. Where previous known tests did not exhibit particuliar bias, these new tests reveal structural, statistical biases for DES, AES, Snow and Lili-128. Other cryptosystems are currently tested and may present unsuspected biases.

These tests are still rather quantitative tests but nonetheless they allow to detect possible structural weaknesses in the output ANFs. Current research focuses on more qualitative test involving factorial experiments. It should provide necessary information to greatly improve previous cryptanalytic techniques.

Acknowledgement

Many thanks to Ralph Wernsdorf (Rohde & Schwarz SIT) who detected a few errors in the simulations and in the results. Thus he helped me to greatly improved them and consequently this paper.

References

1. *http://www.nist.gov/aes/*
2. A. Bosselaers, B. Preenel editors, *Intregrity Primitives for Secure Information Systems: Final Report of RACE Integrity Primitives Evaluation RIPE-RACE 1040*, LNCS 1007, Springer, 1995.
3. H. Dobbertin, A. Bosselaers, B. Preenel, RIPEMD-160: a Strengthened Version of RIPEMD. In. *D. Gollman ed., Fast Software Encryption, Third International Workshop*, LNCS 1039, Springer, 1996.
4. W. Feller, *An Introduction to Probability Theory*, Wiley, 1966.
5. *http://www-rocq.inria.fr/codes/Eric.Filiol/index.html*
6. E.Filiol, New Combinatorial Cryptanalysis Techniques, Private Report, 2002.
7. FIPS 46, *Data Encryption Standard*, Federal Information Processing Standards Publication 140-1, US Dept of Commerce/NIST, 1977.
8. FIPS 140-1, *Security Requirements for Cryptographic Modules*, Federal Information Processing Standards Publication 140-1, US Dept of Commerce/NIST, 1994.
9. FIPS 180, *Secure Hash Standard*, Federal Information Processing Standards Publication 180, US Dept of Commerce/NIST, 1993.
10. FIPS 180-1, *Secure Hash Standard*, Federal Information Processing Standards Publication 180-1, US Dept of Commerce/NIST, 1995.
11. S.W. Golomb, *Shift Register Sequences*, Aegean Park Press, 1982.
12. R.V. Hogg, E.A. Tanis, *Probability and Statistical Inference*, MacMillan, 1988.
13. D.E. Knuth *The Art of Computer Programming*, Vol. 2, Addison Wesley, 1981.
14. P. J. McCarthy. *Introduction to Arithmetical Functions.* Springer, 1986.
15. A.J. Menezes, P.C. Van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1997.
16. J.L. Massey, Shift-Register Synthesis and BCH Decoding, *IEEE Trans. on Inf. Th.*, Vol. IT-15, pp 122–127, 1969.
17. U. Maurer, A Universal Statistical Test for Random Bit Generators, *J. of Cryptology, 5* pp 89-105, 1992.
18. *http://www.cryptonessie.org*
19. D. Olejár, M. Stanek, On Cryptographic Properties of Random Boolean Functions, *Electronic Journal of Universal Computer Science, Vol. 4, Issue 8*, 1998.
20. B. Schneier, *Applied Cryptography*, Wilew et Sons, 2nd ed., 1996.
21. R.L. Rivest, The MD4 Message Digest Algorithm, *Advances in Cryptology - CRYPT0'90*, LNCS 537, Springer, 1991.
22. R.L. Rivest, The MD5 Message Digest Algorithm, Internet Request for Comment 1321, April 1992.
23. T. Siegenthaler, Correlation Immunity of Nonlinear Combining Functions for Cryptographic Applications, *IEEE Trans. on Inf. Th.*, Vol. IT 35, pp 776–780, 1984.
24. G. Xiao, J.L. Massey, A Spectral Characterization of Correlation Immune Functions, *IEEE Trans. on Inf. Th.*, Vol. IT-34, pp 569–571, 1988.
25. Y. Zheng, J. Pieprzyk, J. Seberry, HAVAL - A One-way Hashing Algorithm with Variable Length of Output, *Advances in Cryptology - AUSCRYPT'92*, LNCS 718, Springer, 1993.

Message Authentication Codes with Error Correcting Capabilities

Charles C.Y. Lam[1], Guang Gong[2], and Scott A. Vanstone[1]

[1] Department of Combinatorics & Optimization
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
{cy3lam,savansto}@cacr.uwaterloo.ca

[2] Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
ggong@cacr.uwaterloo.ca

Abstract. In this paper, we propose classes of Message Authentication Codes (MAC) based on error correcting-codes. We introduce a new notion of error tolerant forgery of hash messages. These MACs allow full error recovery for all applications, while being error-tolerant for less information-sensitive applications. The classes of the keyed hash functions are highly secure, and provide the capabilities of correcting errors on transmission, including burst-errors, which is a typical phenomenon in wireless communications. These classes of hash functions are easily implementable in hardware by means of simple linear feedback shift register structures.

Keywords: Message Authentication Codes, Error Correcting Codes

1 Introduction

Keyed hash functions, also known as *message authentication codes* (MAC) in the literature [9], provide security service of data integrity and data origin authentication in network communications. However, due to the characteristics of transmission channels, data may be corrupted by errors occurred during the communication process. In this case, the verification for integrity check of data would not be possible. Thus, it is interesting to construct MACs which are tolerant to a few errors that may occur during transmission. The purpose of this paper is to construct MACs which allow at most 2 errors or one segment of burst error of length n to occur during communication, by means of the well known BCH codes and Reed-Solomon codes [8,10].

Applying error correcting codes to construct MACs was first exploited by Krawczyk [7] in Crypto '94. In that paper, Krawczyk used all distinct Hamming codes of length $2^n - 1$ as a space of keys for the MAC, and the parity check bits as the hash value of a message. He showed that the functions introduced are highly secure with a failure probability of 2^{-30} authenticating 1 Gbit of information.

However, the author did not make use of the error-correcting property that comes with the construction.

In our scenario, we will consider the scheme where the MAC will perform error correcting operations, which we will call an *error correcting MAC*. We will show that this scheme possess high security strength. Note that this MAC will be error-correcting, and unless the messages transmitted is not information sensitive, such a MAC is not *error tolerant*, and an incoming MAC must be error-corrected.

Traditionally, error correcting codes are used to transmit a MAC by first computing the MAC and then adding error correcting codes for transmission. However, due to the way current error correcting codes are applied, the message is first divided into many small blocks, and then error correcting codes are applied, resulting in large message expansion. In our MAC, the error correcting code is embedded into the MAC, and although not capable of correcting the amount of errors that a regular error correcting code can process, can still correct burst errors of a respectable length, without significant message expansion.

The rest of the paper is organised as follows. In Section 2, we give a construction based on BCH codes. We analyse the security in Section 3.1 and the complexity and implementation issues in Section 3.2. In Section 4, we introduce the construction using Reed-Solomon codes, and then discuss the related security, complexity and implementation issues.

To conclude this section, we will give a formal definition of classes of hash functions, and the construction of the proposed MACs given in [7].

Definition 1 *An (m, n)-family H of hash functions over a finite field $\mathbb{F}$ is a collection of functions that map the set of strings of length m over $\mathbb{F}$ into the set of strings of length n over a $\mathbb{F}$.*

For simplicity, we assume a communication scenario in which two parties communicate over an unreliable channel with a malicious adversary in the middle. The secret key used in the communication is unknown to the adversary. The parties exchange using the secret key for only one message of length m, where the secret key is a hash function h drawn randomly from the (m, n)-family of hash functions and a random pad r of length n. Let $\mathbb{Z}_2^n$ be the vector space of dimension n over GF(2). We consider the messages M to be vectors over $\mathbb{Z}_2^m$ and the hash values to be vectors over $\mathbb{Z}_2^n$. The sender sends the message M together with the tag $t = h(M) + r$, and is verified at the receiver by recalculating t.

Construction 1 *[7, Section 3.1] Let $H_{m,n}$ be an (m, n)-family of hash functions over $\mathbb{Z}_2$ defined as follows. Let $f(x)$ be an irreducible polynomial of degree n over* GF(2)*. Let M be a message of predetermined binary length m, and $M(x)$ be the polynomial over* GF(2) *with coefficients corresponding to the bits of M. That is, if $M = (s_{m-1}, s_{m-2}, \ldots, s_0)$, then $M(x) = s_{m-1}x^{m-1} + s_{m-2}x^{m-2} + \cdots + s_1 x + s_0$. We associate the hash function h_f for a message M to be*

$$h_f(M) = M(x) \cdot x^n \bmod f(x).$$

The construction, if considered as a codeword in error-correcting codes context for a fixed irreducible polynomial $f(x)$, is a Hamming code [8,10] of distance 3, thus is able to correct up to 1 error upon transmission.

2 Error Correcting MAC Based on BCH Codes

In this section, we first give a definition for the error correcting MAC, then we give a detailed construction of such a MAC using binary BCH codes with designed distance 5.

Construction 2 *Let $H_{m,l}$ be an (m,l)-family of hash functions over $\mathbb{Z}_2$ defined as follows. Let C be a binary BCH code of length $2^n - 1$ with minimal distance $2e+1$ and $g(x)$ be the parity check polynomial of C of degree l. Let M be a message of predetermined binary length m, and $M(x)$ be the polynomial over* GF(2) *with coefficients corresponding to the bits of M. The hash function h_g associated with the polynomial g is defined as*

$$h_g(M) = M(x) \cdot x^l \bmod g(x),$$

and

$$H_{m,l} = \{h_g \mid \text{all valid parity check polynomials } g\}.$$

Let D be an error decoding algorithm for a code with minimal distance $2e+1$. D can further be decomposed into two sub-procedures: D_1 and D_2 where D_1 is the error detection algorithm (equivalently, the computation of the error locator polynomial), and D_2 is the error correction operation.

Scheme 1 (Error Correcting MAC) *The algorithm is as follows:*
Sender: To send a message M of length m,

Step 1. Select $h_g \in_R H_{m,l}$.
Step 2. Choose a random pad $u \in \mathbb{Z}_2^l$.
Step 3. Compute

$$r(x) = h_g(M) = M(x) \cdot x^l \bmod g(x)$$

and set

$$t(x) = r(x) + u(x).$$

Step 4. Convert $t(x)$ into the corresponding bit string of length l and then send the message-tag pair (M, t).

Receiver: Suppose the received message-tag pair is (M', t').

Step 1. Compute

$$h_g(M') = M'(x) \cdot x^l \bmod g(x).$$

Verify that

$$t'(x) + u(x) = h_g(M') \tag{1}$$

If (1) is true then accept the pair (M', t') as valid (we have $M' = M$ and $t' = t$), in this case no errors occurred during transmission.

Step 2. Otherwise, remove the random pad from t', let

$$r' = t' + u.$$

Apply D_1 to (M', r'). If D_1 returns $e+1$ errors or more, then reject the pair (M', t').

Step 3. If (M', r') has at most e errors, apply D_2 to find the error locations detected in D_1. Let (M'', r'') be the resulting pair, that is,

$$(M'', r'') = D_2 \circ D_1(M', r')$$

where "$\circ$" represents the composition of the two operators D_1 and D_2. If the error location are found to be outside of the length of the message-tag pair, when the word itself is considered to be the codeword of length $2^n - 1$ in the sense of a full BCH codeword, then reject the message, otherwise accept (M'', r'') as the valid message-tag pair.

Definition 2 *Let $d(A, A')$ represent the Hamming distance of two codewords A, A' of the same length. If $A = (M, t)$ is a correct message-tag pair, and $A = (M', t')$ is another message-tag pair of the same length, such that*

$$d(A, A') \leq e,$$

then $A' = (M', t')$ is considered as an equivalent acceptable pair, corresponding to the same message-tag pair $A = (M, t)$.

Under this definition, it seems that we have violated the definition of a secure MAC. However, if we allow error correcting capability for a MAC, such a definition will be necessary. Hence, we do not consider replacing a correct message-tag pair A with an equivalent acceptable message-tag pair A' which is of Hamming distance less than e from A as a message attack, since A' is correctable to A, thus it is considered as the same message.

In the following we will discuss in detail Scheme 1 by using the BCH code with designed distance 5.

Construction 3 *Let $n \geq 7$ be an odd integer, so that* $\gcd(3, 2^n - 1) = 1$. *Let $H_{m,2n}$ be a $(m, 2n)$-family of hash functions defined as follows. Let α be a primitive element in* $\mathrm{GF}(2^n)$ *and $f_\alpha(x)$ be its minimal polynomial of degree n over* $\mathrm{GF}(2)$*, where α is a root over* $\mathrm{GF}(2^n)$. *Let $f_{\alpha^3}(x)$ be the minimal polynomial of degree n for α^3 over* $\mathrm{GF}(2)$*, and let $g_\alpha(x) = f_\alpha(x) f_{\alpha^3}(x)$. The hash function h_g associated with the polynomial g for the message M is defined as*

$$h_{g_\alpha}(M) = M(x) \cdot x^{2n} \bmod g_\alpha(x).$$

Scheme 2 (2-error Correcting MAC) *The algorithm is as follows:*
Sender: For a message M:

Step 1. Select in random a primitive element $\alpha \in$ $\mathrm{GF}(2^n)$ *and compute the minimal polynomials $f_\alpha(x)$ and $f_{\alpha^3}(x)$. Set*

$$g_\alpha(x) = f_\alpha(x) f_{\alpha^3}(x).$$

Step 2. Select $r \in_R \mathbb{Z}_2^{2n}$
Step 3. Compute the hash value of M:

$$h_{g_\alpha}(M) = M(x) \cdot x^{2n} \bmod g_\alpha(x).$$

Receiver: For the message-tag pair (M', t'):

Step 1. Form the polynomial

$$s(x) = M'(x) \cdot x^{2n} + t'(x) + r(x).$$

Compute

$$A_1 = s(\alpha),$$
$$A_3 = s(\alpha^3).$$

The receiver will accept (M', t') if $A_1 = A_3 = 0$. There are no errors occurred during transmission.
Step 2. Otherwise errors may have occurred. If $A_1 \neq 0$ and $A_3 = A_1^3$, we set the error locator polynomial as

$$\sigma(z) = 1 + A_1 z.$$

Else we set

$$\sigma(z) = 1 + A_1 z + (\frac{A_3}{A_1} + A_1^2) z^2.$$

Step 3. Solve the error locator polynomial for roots z_1 (and z_2, depending on the degree of the polynomial). Let $x_i = z_i^{-1}$. The discrete logarithm of x_i with respect to α are the locations of the errors. If the error locations are out of range then we reject the message-tag pair. Otherwise let E be the error vector and let

$$M'' = M' + E,$$

and accept the message-tag pair as valid with corrected message M''.

Remark: Note that the random pad $r \in_R \mathbb{Z}_2^{2n}$ is needed. Otherwise, a simple attack by taking the polynomial factorization $s(x)$ of a correctly transmitted s will likely lead to the factors of $g_\alpha(x)$.

Remark: Although the MAC proposed includes the parameter of the length of the message, it is noted that the length m is only used in security analysis and decoding while the hash functions are only dependent on the polynomial $g_\alpha(x)$. The length, however, is to be noted during transmission, for the reason of error correction and security. We will discuss this issue in the following section.

Remark: Solving the discrete logarithm of x_i seems to be a computationally infeasible operation in the field GF(2^n) for large n. However, since the length of the received word (M, t) is known, calculating the discrete logarithm over a much smaller bounded continuous region requires much less work. We will discuss the complexity of the MAC generation, verification, and error correction in Section 3.2.

Remark: Note that our construction applies to BCH codes of larger distances. For example, in literature, there are two types of BCH codes which give a minimal distance of 7. We can take [8,10] the polynomial with designed distance 7, where

$$g(x) = f_\alpha(x) f_{\alpha^3}(x) f_{\alpha^5}(x),$$

or to take [3]

$$g(x) = f_\alpha(x) f_{\alpha^d}(x) f_{\alpha^{d^2}}(x),$$

where n is odd and $d = 1 + 2^{\frac{n-1}{2}}$.

3 Security and Complexity Analysis of Error Correcting MAC through BCH Codes of Distance 5

Using the construction through BCH codes, we will show the security aspects of the class $H_{m,2n}$. The following is given by Krawczyk in [7]. The security analysis for general BCH codes is similar. The complexity and implementation issues follow.

3.1 Security Analysis

Definition 3 *A family of functions $H_{m,l}$ over $\mathbb{F}$ is linear if for all $M, M' \in \mathbb{F}^m$ we have $h(M+M') = h(M)+h(M')$, where M, $M' \in \mathbb{F}^m$ and $h(M)$, $h(M') \in \mathbb{F}^l$.*

Definition 4 *A family $H_{m,l}$ of hash functions over $\mathbb{F}$ is called ϵ-balanced if for all $M \neq 0, M \in \mathbb{F}^m$, $c \in \mathbb{F}^l$*

$$\Pr_h(h(M) = c) \leq \epsilon.$$

It can be easily shown that our class of hash functions $H_{m,2n}$ over $\mathbb{Z}_2$ for any fixed m and n is linear and ϵ-balanced. Let $w(\cdot)$ represent the hamming weight of a string.

Because of the ability of error correcting, we consider any vector of length $m + l$ within hamming distance e from a correct message-tag pair (M, t) to be in the same *equivalence class*. That is, the messages within distance e of (M, t) are all acceptable message-tag pairs corresponding to the same correct message (M, t).

Definition 5 *A hash function $h_g \in_R H_{m,l}$ over $\mathbb{F}$ is e-error-forgery-resistant with probability ϵ if and only if given any message-tag pair within distance e of a correct pair (M, t), where $M \in \mathbb{F}^m$, $t = h_g(M) + r \in \mathbb{F}_l$, $r \in_R \mathbb{F}^l$, no adversary succeeds by finding some M' of length m and t' of length l, with $M' \neq M$, $d(M', M) > e$ such that (M', t') is an acceptable pair with probability larger than ϵ.*

Theorem 1 *A family $H_{m,l}$ of hash functions is e-error-forgery-resistant with probability ϵ if and only if*

$$\forall c, M_1 \not\exists M_2, d(M_1, M_2) > e, \sum_{e' \in \mathbb{F}^l, w(e') \leq e} \Pr_h(h(M_1) + h(M_2) = c + e') \leq \epsilon.$$

The proof of the above statement is similar as in Theorem 5 in [7], except we allow errors to be introduced in the transmitted word.

Proof. We consider all the vectors within distance e from a correct message-tag pair (M, t) to be in the same equivalent class. The pair (M_1, t_1) is successfully replaced by the pair (M_2, t_2) if and only if for the secret function h and the random vector r we have $t_1' = h(M_1') + r$ and $t_2 = h(M_2) + r + e'$, where M_1' and t_1' are the error-free message-tag pair associated with M_1 and t_1, and $e' \in \mathbb{F}^l$ with $0 \leq w(e') \leq e$. Note that the error e' already includes the possibility of having errors in the message part. Equivalently, we have $c = t_1' + t_2 + e' = h(M_1' + M_2)$. Hence, the probability of success for an adversary is bounded by

$$p = \sum_{e' \in \mathbb{F}^l, w(e') \leq e} \max_{M_1, M_2, c} \Pr_h(h(M_1) + h(M_2) \in \mathcal{C}_c),$$

where $\mathcal{C}_c$ is the equivalent class of vectors within distance e of c. Notice that this success probability is achievable whenever the transmitted message is one of the messages in which the maximum is attained, by replacing (M_1', t_1') by $(M_2, t_1' + c)$. Since the correct words (M, t) are distance at least e apart, we must have

$$p \leq \epsilon.$$

Similar to [7], we can draw the following consequence.

Theorem 2 *If $H_{m,l}$ is linear then $H_{m,l}$ is e-error-forgery-resistant with probability ϵ if and only if $H_{m,l}$ is ϵ-balanced.*

Theorem 3 *For any values of m and n, the family of hash functions $H_{m,2n}$ in Construction 3 is linear and ϵ-balanced for*

$$\epsilon \leq \frac{6(m + 2n) \ln \ln(2^n - 1)}{2^n - 1},$$

and 2-error-forgery-resistant with probability ϵ.

Proof. The family of hash functions is obviously linear. Now given any polynomial $g_\alpha(x) = f_\alpha(x) f_{\alpha^3}(x)$ where α is a primitive element in GF(2^n), any non-zero message M of length m and any string c of length n,

$$\begin{aligned} h_g(M) = c &\iff M(x) \cdot x^{2n} \bmod g_\alpha(x) = c(x) \\ &\iff g_\alpha(x) | (M(x) \cdot x^{2n} - c(x)). \end{aligned}$$

Note that $q(x) = M(x) \cdot x^{2n} - c(x)$ is a non-zero polynomial of degree at most $m+2n$. Thus there are at most $\frac{m+2n}{n}$ factors of primitive polynomials of degree n in $q(x)$. Therefore, there are at most $\frac{m+2n}{n}$ possible hash function candidates. Let $\phi(\cdot)$ be the Euler Phi function. While there are $\frac{\phi(2^n-1)}{n}$ possible hash functions in this family, we have the probability

$$\Pr(h_g(M) = c) \leq \frac{(m+2n)/n}{\phi(2^n-1)/n} = \frac{m+2n}{\phi(2^n-1)}.$$

Using the estimate

$$\phi(2^n-1) > \frac{2^n-1}{6 \ln\ln(2^n-1)},$$

the result follows. Note that if the received message-tag pair is error-prone within distance 2 of a correct code word, the factorisation of $q(x)$ will not contain all of the factors of $g_\alpha(x)$, thus lowering the probability value ϵ.

For $\epsilon < 2^{-80}$, $n \geq 93$ is required. Below is a table for the values of m and n which guarantee a probability $\epsilon < 2^{-80}$.

	$\epsilon < 2^{-80}$
$n = 100$	$m < 41031$
$n = 120$	$m < 4.14 \times 10^{10}$
$n = 140$	$m < 4.19 \times 10^{16}$

For $\epsilon < 2^{-160}$, $n \geq 174$ is required. Below is a table for the values of m and n which guarantee a probability $\epsilon < 2^{-160}$.

	$\epsilon < 2^{-160}$
$n = 180$	$m < 35849$
$n = 200$	$m < 3.71 \times 10^{10}$
$n = 220$	$m < 3.82 \times 10^{16}$

In practice, two communicating parties would exchange multiple messages, one particular hash function h_g can be used for the multiple messages, as long as a different random pad r is used for each message. In fact, since a random pad is applied, it is useless that the adversary sees many message-tag pairs.

In the practice of error correcting codes, the error correcting property also mean the capability to be able to *detect* errors without correcting. However, in this MAC, the procedure of error correcting *must* be applied. The reason being that the security of the MAC is dependent on the length of the message-tag pair. If error detection is done without error-correcting, this would allow a message of a different length (typically much longer one) of short Hamming distance to be accepted as a valid message-tag pair. In other words, it is highly likely that the any pair $(M,t) \in \mathbb{Z}_2^{m+2n}$ such that when considered over a codeword of length $2^n - 1$ under the normal sense of the BCH codeword $C_1 \in \mathbb{Z}_2^{2^n-1}$, where

$$C_1 = (\underbrace{0, 0, \cdots 0}_{2^n-1-m-2n}, \underbrace{M, t}_{m+2n})$$

is within distance 2 of a codeword such as

$$C_2 = (\underbrace{0, \cdots 0, \times, 0, \cdots 0, \times, 0, \cdots 0}_{2^n-1-m-2n}, \underbrace{M, t}_{m+2n}) \in \mathbb{Z}_2^{2^n-1}.$$

However, the restriction of the length of the message limits the probability (defined by ϵ) that such an attack can happen. This also illustrates the reason for the upper limit on the length m of the message for any fixed ϵ and n.

Remark: In Construction 3, we propose to use $g_\alpha(x) = f_\alpha(x) f_{\alpha^3}(x)$. There are in fact more choices $g(x) = f_\alpha(x) f_{\alpha^k}(x)$ which give a BCH code of minimal distance at least 5. While there are no results on which values of k or the proportion of such polynomials that give codes of distance at least 5, any such results would greatly improve the probability value ϵ, and thus increase the possible number of hash functions in the class.

3.2 Complexity and Implementation

In a typical communication session, such a MAC algorithm requires a predetermined choice of a primitive element $\alpha \in \mathrm{GF}(2^n)$, together with a pseudorandom bit string r of length $2n$, which together forms our secret key. The polynomial $g_\alpha(x)$ is therefore $f_\alpha(x) f_{\alpha^3}(x)$. The message-tag pair (m, t) is then computed and sent over an error-prone channel. The receiver calculates the tag using $g_\alpha(x)$. If the tag does not match, the receiver may first attempt to error correct the message-tag pair by considering $(m, t + r)$ as a codeword with limited length $m + 2n$. If the error correcting is successful, the receiver will accept the message-tag pair. Note that if the error location is out of the range of the $m + 2n$ bits, then the error correction fails, thus the receiver may reject the message-tag pair.

The generation of the message-tag pair requires a multiplication by x^{2n} followed by an operation of the division algorithm. It is easy to see that the division algorithm takes $2mn$ bit operations over GF(2). The implementation of this operation is simple over software. In hardware, such an operation can be implemented through a linear feedback shift register (LFSR) with architecture allowing input of a variable $g_\alpha(x)$.

The more computationally heavy operation is the error-correcting part. We are required to first solve a quadratic equation over $\mathrm{GF}(2^n)$ and then compute at most two discrete logarithms of type $x = \log_\alpha \beta$ where α is a primitive element of $\mathrm{GF}(2^n)$ (specified by $g_\alpha(x)$) and $0 \le x \le m + 2n - 1$ (recall that if x is outside of this range then we reject the message). Finding the roots of a quadratic equation is easy with an algorithm outlined in IEEE P1363 [5]. As for the part of solving the discrete logarithm problem, an exhaustive search for the discrete logarithms for a short length message-tag pair is feasible as such an operation requires only $m + 2n$ multiplication operations in $\mathrm{GF}(2^n)$, which is relatively fast by choosing an appropriate basis representation. If, say, a baby-step-giant-step algorithm is used for decoding on a device with abundant memory storage, such an operation requires $\sqrt{m + 2n}$ storage of group elements, $2\sqrt{m + 2n}$ multiplications, and at most $\sqrt{m + 2n}$ table lookups.

The key size of the hash functions in the use as a MAC is $3n$, including n bits for the element α and $2n$ bits for the pseudorandom pad r.

There are other possible implementations on MAC with error correcting capabilities. One method is to apply some other MAC algorithms to get the message-tag pair, and append the result with error correcting codes. In this case the industrial codes are usually on message segments of short lengths. We will end up with a transmitted message of longer length but corrects more errors. The length of the transmitted message will be equal to the length of the message, plus the length of the MAC, and the length of the error correcting bits. The information rate of such an implementation is dependent on the error correcting code we choose.

In our case, the MAC itself is also the syndrome for error correction. Hence the information rate is at least $\frac{m}{m+2n}$. The message we transmit is shorter in the sense that the MAC itself is also the syndrome for error correction.

4 Error Correcting MAC Based on Reed-Solomon Codes

In this section, we give another construction by applying Reed-Solomon Codes. Let $\alpha \in \mathrm{GF}(2^n)$. A Reed-Solomon code with designed distance δ has generator polynomial

$$g(x) = (x - \alpha^b)(x - \alpha^{b+1}) \cdots (x - \alpha^{b+\delta-2}).$$

The resulting code has a distance of at least δ over $\mathrm{GF}(2^n)$.

4.1 Construction

We can map a binary message of length $m = kn$ where $k \in \mathbb{Z}$ into a polynomial over $\mathrm{GF}(2^n)$ by dividing up the message into n-bit blocks and consider each as an element in $\mathrm{GF}(2^n)$. For example, if a polynomial basis is used, then each bit in the n-bit block can be considered as the coefficient to the linear combination of the ordered basis elements, and the whole n-bit block is one element represented by the linear combination. Padding is necessary if the message length is not a multiple of n. However, caution must be taken in the method for padding.

Construction 4 *Let $m = kn$ and $H_{k,\delta-1}$ be a family of hash functions defined over $\mathrm{GF}(2^n)$ as follows. Let α be a primitive element in $\mathrm{GF}(2^n)$. Let $1 \leq b \leq 2^n - 1$ where $\gcd(b, 2^n - 1) = 1$. Let $\delta = 2e + 1$ for some positive integer e. Let*

$$g_{\alpha,b,\delta}(x) = (x - \alpha^b)(x - \alpha^{b+1}) \cdots (x - \alpha^{b+\delta-2}).$$

Let $M(x)$ be the polynomial over $\mathrm{GF}(2^n)$ of degree k associated with the message M of binary length m by a bijective mapping as described above. We associate a hash function h_g for the message M to be

$$h_g(M) = M(x) \cdot x^{\delta-1} \bmod g_{\alpha,b,\delta}.$$

The above construction leads us to a hash function capable of correcting *burst errors*. We can consider the message-tag pair not only as a codeword over $\mathrm{GF}(2^n)$ but also over $\mathbb{Z}_2$ by considering each element represented over a certain basis and the coefficients over $\mathbb{Z}_2$ of the linear combination as message bits (using the same bijective mapping as described). The result is an $H_{m,(\delta-1)n}$ family of hash functions. The following is a standard result from coding theory.

Definition 6 *A cyclic burst of length t is a vector whose non-zero components are within t successive entries (cyclically), and which the first and the last entry are non-zero.*

Fact 1 *The resulting message-tag pairs (M, t) from Construction 4 is capable of correcting up to e errors when considered as a word over $\mathrm{GF}(2^n)$, and correcting a burst error of length up to $n(e-1)+1$ when considered as a word over $\mathbb{Z}_2$.*

Definition 7 *A hash function is e-burst-error-forgery-resistant with probability ϵ if and only if given any message-tag pair within a burst error of length e of a correct pair (M, t), where M is of length m, $t = h_g(M) + r$ of length $(\delta-1)n$, $h_g \in_R H_{m,(\delta-1)n}$, $r \in_R \mathbb{Z}_2^{(\delta-1)n}$, no adversary succeeds by finding some M' of length m and t' of length $(\delta-1)n$, with $M' \neq M$, and $(M', t') - (M, t)$ being a burst error vector of length $> e$, such that (M', t') is an acceptable pair with probability larger than ϵ.*

4.2 Security Analysis

Theorem 4 *The family of hash functions $H_{k,n}$ in Construction 4 is*

1. *linear*
2. *ϵ-balanced where*
$$\epsilon \leq \frac{\binom{k+\delta-1}{\delta-1}}{(\phi(2^n-1))^2}.$$
3. *e-error-forgery-resistant over $\mathrm{GF}(2^n)$ with probability ϵ.*
4. *$n(e-1)+1$-burst-error-forgery-resistant over $\mathbb{Z}_2$ with probability ϵ (when considered as the family $H_{kn,n(\delta-1)}$).*

Proof. The proof is similar to Theorem 3 except now that

$$q(x) = M(x) \cdot x^{\delta-1} - c(x)$$

may split into linear factors. Since our choice of α can be any primitive element in $\mathrm{GF}(2^n)$ and the choice of b is any integer relatively prime to $2^n - 1$, we have $(\phi(2^n-1))^2$ different functions with respect to any given δ.

Below is a table for the values of n and $k = \frac{m}{n}$ which guarantee a probability $\epsilon < 2^{-80}$.

	$\delta = 5$	$\delta = 7$	$\delta = 9$
$n = 80$	$k < 1.52 \times 10^6$	$k < 23301$	$k < 3111$
$n = 120$	$k < 1.39 \times 10^{12}$	$k < 2.20 \times 10^8$	$k < 2.99 \times 10^6$
$n = 160$	$k < 1.67 \times 10^{18}$	$k < 2.48 \times 10^{12}$	$k < 3.27 \times 10^9$

4.3 Complexity and Implementation Issues

The implementation of decoding Reed-Solomon codes have been discussed thoroughly in literature [1,2,8,10,6,12].

Due to the similar reason as in the MACs using BCH codes, it is necessary to compute the error locations during decoding. However, for applications to less information-sensitive transmissions, such as voice and image signals, a few errors occurred during transmission is not crucial. Therefore, the step to correct the errors may not be required and we can simply take the received message M as the accepted message if error locations are found to be within the range of the length of M.

The major computation problem here we are faced with, again, is the solution of the discrete logarithm with base α over a limited region of length $k+\delta-1$ for the location of error. The computational aspect is similar as in the BCH case.

However, if only burst errors are corrected, there is a much simpler algorithm which requires only $k+\delta-1$ clock cycles of operations. This is discussed in the following section.

4.4 Error-Trapping Method for Burst Errors Decoding

The method is described in [10], here we give a brief explanation of the algorithm.

Let $e(x)$ be the error polynomial. Let

$$s_j(x) \equiv x^{-j} e(x) \pmod{g_{\alpha,b,\delta}(x)}.$$

If $s_j(x)$ is a polynomial of burst length e or less, then

$$e(x) = x^j s_j(x)$$

is the error polynomial.

The implementation of this decoding method is done by calculating

$$s_j(x) \equiv x^{-j} r(x) \pmod{g_{\alpha,b,\delta}(x)}$$

starting from $j = 0$. Note that for each $j \geq 1$, $s_j(x)$ can be obtained from a shift operation of the polynomial from $s_{j-1}(x)$, plus an addition of a polynomial over $\mathrm{GF}(2^n)[x]$ of degree δ (which is the inverse of x with respect to $g_{\alpha,b,\delta}(x)$). We stop when there are no such bursts within $s_0(x), \ldots, s_{k+\delta-1}(x)$. Hence this operation takes $k+\delta-1$ clock cycles of simple XOR operations in hardware.

5 Conclusion

We have provided an error correcting MAC which can provide integrity check for corrupted data which corrects at most 2 errors or an $n(e-1)+1$-bit burst error during transmission. In the process, we introduced the notion of error-tolerant forgery of hash messages. The security of such MACs remain high, and can be applied to less information-sensitive wireless transmissions such as voice communications. It remains open research problems to increase the size of key space, to efficiently decode BCH and Reed-Solomon codes over a large field, and to solve the discrete logarithm problem over a short continuous interval. It is also desirable to find classes of error correcting codes that locate errors without having to solve the discrete logarithm problem.

Acknowledgement

The authors wish to thank the referees for the comments on information rate issues.

References

1. E.R. Berlekamp, *Algebraic coding theory*, McGraw-Hill, New York, 1968.
2. R.E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Co., May 1984.
3. A. Chang, P. Gaal, S.W. Golomb, G. Gong, T. Helleseth, P.V. Kumar, *On a Conjectured Ideal Autocorrelation Sequence and a Related Triple-Error Correcting Cyclic Code*, IEEE Trans. Information Theory, Vol. 46, No. 2, March 2000.
4. S.W. Golomb, *Shift Register Sequences,* Holden-Day, Inc., 1967. Revised edition, Aegean Park Press, 1982.
5. IEEE P1363, *Standard Specifications For Public Key Cryptography*, 2000.
6. J.H. Jeng and T.K. Truong, *on Decoding of Both Errors and Erasures of a Reed-Solomon Code Using an Inverse-Free Berlekamp-Massey Algorithm*, IEEE Transactions on Communications, Vol. 47, No. 10, October 1999, 1488-1494.
7. H. Krawczyk, *LFSR-based Hashing and Authentication*, Advances in Cryptology–Crypto '94, Lecture Notes in Computer Science, LNCS 839(1994), Springer-Verlag, 129-139.
8. F.J. MacWilliams, N.J.A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Publishing Company, 1977.
9. A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
10. S.A. Vanstone, P.C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic Publishers, 1989.
11. E. Bach, J. Shallit, *Algorithmic Number Theory*, MIT Press, Cambridge, Massachusetts, 1996.
12. T.K. Truong, J.H. Jeng, I.S. Reed, *Fast Algorithm for Computing the Roots of Error Locator Polynomials up to Degree* 11 *in Reed-Solomon Decoders*, IEEE Transactions on Communications, Vol. 49, No.5, May 2001, 779-783.

The Consistency of an Access Control List*

Shou-peng Li[1], Shi-zhong Wu[2], and Tao Guo[3]

[1] Department of Mathematics, Sichuan University, Chengdu, China
Yanlu99@21cn.com
[2] Department of Mathematics, Sichuan University, Chengdu, China
director@cnistec.gov.cn
[3] Department of Computer Science & Technology,
Huazhong University of Science and Technology, Wuhan, China
gttong@sina.com

Abstract. This paper gives firstly an introduction about the consistency of an Access Control List (ACL). It uses triples or in other words, in three dimensions and their corresponding decisions to describe the access control rules, i.e. ACL entries. We present then and justify the mathematical definitions of an ACL and its related components and discuss its consistency, two theorems about the consistency of the ACL. Thirdly the paper discusses verification of the consistency of the ACL and a new method used to verify the consistency of the ACL derived from the theorems in use. The method obtained here is a theoretical approach. In the conclusion, we point out that in a practical system, verification of the consistency of the ACL is necessary and the method is an effective and efficient one, further measures should be taken and study should be continued to reduce the amount of computation needed to examine the consistency of the ACL.

Keywords: access control, consistency, information security

1 Introduction

The goal of information security is to attain confidentiality, integrity and availability. Access control is required to achieve these three objectives. There are several approaches for implementing the access matrix model in computer system, the most popular of which is Access Control List, names ACL ([1]). The conventional concept of an ACL is the architectural foundation of many authorization mechanisms. A typi-

* Supported by the National Natural Science Foundation of China under Grant No.90104033.

LI Shou-peng was born in 1966. He is the director of the InfoSec Laboratory of China Information Technology Security Certification Body. He is a Ph.D. Candidate at Department of Mathematics, Sichuan University. His current research interests include information security and cryptography. **WU Shi-zhong** is the director of the China Information Technology Security Certification Body. He is a Ph.D. Candidate at Department of Mathematics, Sichuan University. **GUO Tao** is a Ph.D. Candidate at Department of Computer Science & Technology, Huazhong University of Science and Technology. His current research interests include secure electronic payment, information security and cryptography.

cal ACL is associated with an object to be protected and enumerates the list of authorized users and their rights to access the object. Access rights are selected from a predefined fixed set built into the authorization mechanism. Specification of the subjects is bound to the particular security mechanism employed by the system ([2]).

Here we refer to subjects as active entities in the system, and objects as passive entities. Examples of subjects are processes, and examples of objects are files. Different systems and their implementations will have different ACL structures, representations and features, but in the whole, an ACL always accurately reflects the aspects of the organization's security policy relating to the control of access to the information being protected by the device or system.

The consistency of an ACL means that, there shouldn't be any conflicting entries in the ACL that define contradictive access attributes (access rights).

Another consideration is that the propagation of access right that might occur due to the specific definition of the ACL, even in a consistently defined ACL. This paper is concentrated on the consistency of the ACL. The propagation of access rights is out of the scope of this paper.

In an operating system (OS), the boundary of objects is usually restricted to the passive entities which are easy to be defined and described. It is convenient to enumerate one or a set of objects in a general purpose OS. For example we can designate a file as a single object (exactly object container) and a group of files in a hierarchy (directory) as a set of objects. According to the design, each item in the access control matrix usually stored with its relating object or set of objects. Similarly the boundary of subjects in an OS is also easy to define. It is often easy for us to know all of the users that will have interactions with the system and all of the processes that might act on the users' behalf. After this unambiguous definition of objects and subjects, we can clearly define access rules using the access attribute, such as Read, Write, Execute and others. If we, at first, set a criteria for the possibility of occurrence of conflicting items in the ACL for objects in different level of the hierarchy (i.e. directory and subdirectory) and for subjects in single or collective forms. Then the consistency of the ACL defined above will not be a serious problem,

In a firewall system, the concept of subject and object will have a different meaning than that of an OS. A firewall is always located on the boundary of two adjacent networks, and is the only path of the information flow. The two adjacent networks are usually referred to internal and external networks ([3]). This time, the subjects of the system can be defined less precisely as hosts, group of hosts, sub-nets, clients, processes, applications, users and so on. More strictly, all of them are sets of processes running on the hosts in the separated networks. The objects can also be defined as hosts, group of hosts, sub-nets, services, files and so forth. Similarly, all of them can be abstracted to information contained in containers (files) or services. As the boundaries of subjects and objects in a network system protected by a firewall are hard to be unambiguously defined. This often results in inconsistency in the definition of an ACL with many entries, although the proper operation of the firewall requires that the access control list appropriately reflect the security policy and the ACL should be consistent. A simple example can show the potential existence of inconsistency in the ACL of a firewall. Assume that there is a rule in the ACL stating that host A, in the external network, whose IP address is 202.92.10.100 can fully access host B, in the internal network, whose IP address is 202.93.2.10 and there exist an other rule in the ACL that states that sub-net C, whose IP address is 202.92.10.*, can only use the FTP services on sub-net D, whose IP address is 202.93.*.*, then the two rules above will conflict. In a complex and practical network environment where a firewall may contain hundreds of rules or more, with each rule states an access right in different granularity, inconsistency of a firewall's ACL is often encountered.

In order to have a deep study on the problem of the consistency in an ACL, we'll use a most general abstraction here. Each rule of ACL can be defined as a triple $\mathrm{T} = (\mathrm{S}, \mathrm{O}, \mathrm{A})$ together with a desired decision d. Actually, it can be expanded to n dimension, we skip it for simplicity here.

The triple T represents that a set of subjects S hopes to perform a set of operations A on a set of objects O. We use d as the decision (such as permit, reject, drop and redirect) made by the system upon the triple T, then an access control feature is a function implemented by the system. That is:

$$\mathrm{d} = f(\mathrm{T}) = f(\mathrm{S}, \mathrm{O}, \mathrm{A})$$

Wherein S is a set of subjects, O is a set of objects and A is a set of access attributes.

An access control list can be recognized as a table that defines f. It isn't include some other restrict condition such as the time of access, the location and so on for space limiting, but these factors will never affect the analyzing.

Upon the general abstraction of the access control rules, the consistency among these rules in an ACL can be discussed in the following parts of the paper.

2 The Mathematical Representation ([4])

Let $S = \{s_1, s_2, \cdots, s_m\}$ denote the set of all subjects, S_i as a subset of S, and S^* as the power set of S, i.e.:

$$\forall S_i \in S^* (i = 1,2,\cdots,2^m), \text{ then } S_i \subseteq S$$

Let $O = \{o_1, o_2, \cdots, o_n\}$ denote the set of all objects, O_j as a subset of O, and O^* as the power set of O, i.e.:

$$\forall O_j \in O^* (j = 1,2,\cdots,2^n), \text{ then } O_j \subseteq O$$

Define $A = \{a_1, a_2, \cdots, a_l\}$ represent the set of all kinds of possible accesses or operations that might be taken on objects by subjects. A_k is a subset of A, and A^* is the power set of A, i.e.:

$$\forall \mathrm{A_k} \in A^* (k = 1,2,\cdots,2^l), \text{ then } A_k \subseteq A$$

Let $D = \{d_1, d_2, \cdots, d_h\}$ represent the set of different decisions that might be made by the security mechanism provided by the system according to the security policy.

Thus an ACL is a function from V to D, i.e.

$$f : V \to D, \text{ where } V \subset S^* \times O^* \times A^*$$

The definition of consistency: There shouldn't be two or more entries in the ACL that specify contradicting decisions.

That is, given $\forall V_i = (S,O,A) \in V$, $\forall V_j = (S',O',A') \in V$, if $\exists s,o,a$, thus $s \in S$ and $s \in S'$, $o \in O$ and $o \in O'$, $a \in A$ and $a \in A'$, meaning that the two triples V_i and V_j will intersect, so $d = f(V_i) = f(S,O,A)$ and $d' = f(V_j) = f(S',O',A')$ should be consistent.

For example, in a typical firewall, we can usually assume that $D = \{pass, fail\}$, if there are two entries in an ACL whose (S,O,A), the triples, intersect, then the decision should be pass (or fail) for both of them, and they can't be both a pass and a fail.

For the consistency of the ACL, we have the following theorem.

Theorem 1: An ACL is consistenct, if $\forall V_i, V_j \in V$, where $V_i = (S,O,A)$, $V_j = (S',O',A')$, the following condition holds.

$S \cap S' = \Phi$ or
$O \cap O' = \Phi$ or
$A \cap A' = \Phi$

where Φ denotes null set.

Justification: Let $d = f(V_i) = f(S,O,A), d' = f(V_j) = f(S',O',A')$, so d and d' are the corresponding decisions of two entries in an ACL. Assume $S \cap S' = \Phi$, then the two triples $V_i = (S,O,A)$ and $V_j = (S', O', A')$ described above will not intersect. Thus according to the definition, no matter what values d and d' take, the ACL is consistent. Similarly, this can also be justified when $O \cap O' = \Phi$ or $A \cap A' = \Phi$.

Theorem 2: For $\forall V_i, V_j \in V$, where $V_i = (S,O,A)$, $V_j = (S',O',A')$, and $f(V_i) = d$, $f(V_j) = d'$, $d, d' \in D$, if:

$S \cap S' \neq \Phi$ and
$O \cap O' \neq \Phi$ and
$A \cap A' \neq \Phi$

Then the ACL is consistent if and only if $(d \ R \ d')$, where R is a relation on D representing that the effects of the two decisions will not contradict each other.

Justification: Through the condition given by the theorem, we can easily notice that two triples $V_i = (S,O,A)$ and $V_j = (S',O',A')$ corresponding to the two entries of the ACL will intersect. That is, $\exists$ s,o,a , such that $s \in S$ and $s \in S'$, $o \in O$ and $o \in O'$, $a \in A$ and $a \in A'$. If d and d' is exclusive, i.e. $(d,d') \notin R$, then for s,o,a, there will exist two contrary decisions, as is the necessity.

If $(d\ R\ d')$, then for all $\forall s,o,a$, if $s \in S$ and $s \in S'$, $o \in O$ and $o \in O'$, $a \in A$ and $a \in A'$, there won't be any inconsistent decisions appeared in the entries of the ACL, and the sufficiency is justified.

3 The Consistency Examining

Assume that there exist an ACL that has n entries. An entry in the ACL can be denoted as (S_i, O_i, A_i, d_i). Below we will construct four matrixes M_S, M_O, M_A, M_d.

$$M_S = \begin{pmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \phi_{n1} & \phi_{n2} & \cdots & \phi_{nn} \end{pmatrix} \qquad M_O = \begin{pmatrix} \psi_{11} & \psi_{12} & \cdots & \psi_{1n} \\ \psi_{21} & \psi_{22} & \cdots & \psi_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \psi_{n1} & \psi_{n2} & \cdots & \psi_{nn} \end{pmatrix}$$

$$M_A = \begin{pmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1n} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \varphi_{n1} & \varphi_{n2} & \cdots & \varphi_{nn} \end{pmatrix} \qquad M_d = \begin{pmatrix} \theta_{11} & \theta_{12} & \cdots & \theta_{1n} \\ \theta_{21} & \theta_{22} & \cdots & \theta_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \theta_{n1} & \theta_{n2} & \cdots & \theta_{nn} \end{pmatrix}$$

where:

$$\phi_{ij} = \begin{cases} 1, & \mathrm{S_i} \cap \mathrm{S_j} \neq \Phi \\ 0, & \mathrm{S_i} \cap \mathrm{S_j} = \Phi \end{cases}$$

$$\psi_{ij} = \begin{cases} 1, & \mathrm{O_i} \cap \mathrm{O_j} \neq \Phi \\ 0, & \mathrm{O_i} \cap \mathrm{O_j} = \Phi \end{cases}$$

$$\varphi_{ij} = \begin{cases} 1, & \mathrm{A_i} \cap \mathrm{A_j} \neq \Phi \\ 0, & \mathrm{A_i} \cap \mathrm{A_j} = \Phi \end{cases}$$

$$\theta_{ij} = \begin{cases} 1, & \mathrm{d}_i \boldsymbol{R} \mathrm{d}_j \\ 0, & \mathrm{d}_i \overline{\boldsymbol{R}} \mathrm{d}_j \end{cases} \quad (\boldsymbol{R} \text{ is the relation on } \boldsymbol{D} \text{ defined above})$$

Now we can define an operation on the matrixes.

Let $M_V = M_S \overline{\otimes} M_O \overline{\otimes} M_A = (\phi_{ij} \otimes \psi_{ij} \otimes \varphi_{ij}) = (\alpha_{ij})$, whereas the operation "$\otimes$" is to calculate the boolean product of two binary integers, and "$\overline{\otimes}$" represents to perform element by element the boolean products of each pair of the elements in the same row and the same column of two matrixes and gain a new matrix.

It's easy for us to prove that, if $M_V \overline{\otimes} M_d = (\alpha_{ij} \otimes \theta_{ij}) = M_V$, then the ACL is consistent. This is because of the following:

First, for any $\alpha_{ij} = 1$, through theorem 2 we know that in order to attain consistency of the two rules (rule No. i and j), we need to assured that $(\mathrm{d}_i, \mathrm{d}_j) \in \boldsymbol{R}$, i.e. $\theta_{ij} = 1$, so the boolean product $\alpha_{ij} \otimes \theta_{ij} = 1 = \alpha_{ij}$.

Second, according to theorem 1, we learn that for any $\alpha_{ij} = 0$, the two rules (rule No. i and j), will never contradict each other, whether $\mathrm{d}_i \boldsymbol{R} \mathrm{d}_j$ or $\mathrm{d}_i \overline{\boldsymbol{R}} \mathrm{d}_j$ i.e. there is no restrict to the value of θ_{ij}, it can either be "1" or "0". This time again we get $\alpha_{ij} \otimes \theta_{ij} = 0 = \alpha_{ij}$.

Thus the statement above is demonstrated.

4 Conclusion

This paper discusses a new method to correctly verify the consistency of an ACL. When an ACL that contains many entries is used in a practical system, this method of verification need a large amount of computation, even considering that $\phi_{ij} = \phi_{ji}$ (so are $\psi_{ij} = \psi_{ji}, \varphi_{ij} = \varphi_{ji}$ and $\theta_{ij} = \theta_{ji}$), especially for M_S and M_O. Calculating each of the matrixes M_S, M_O, M_A, M_d, will need to perform $n(n-1)/2 \approx n^2/2$ times operations which are mainly based on the checking of whether two sets intersect or not.

Fortunately, it is unnecessary to execute all these operations during each checking of consistency of ACL in practice. The configuration of secure rules is an evolutionary process. So the consistency checking is on from the beginning. Every time when adding a piece of rule, what the system has to do is to execute the operation between the new rule and the old rules which have been consistent. The number of operation concerning every matrix is just $(n-1)$ (n is the number of current rules). In addition, because of the special property of "$\otimes$" operation, it is not all the entries that have to be computed for matrix M_S, M_O, M_A, M_d and M_v. If $\theta_{ij} = 0$, then it does not need to compute $\alpha_{ij}, \psi_{ij}, \varphi_{ij}, \phi_{ij}$. If $\theta_{ij} \neq 0$ but $\phi_{ij} = 0$, then $\alpha_{ij} = 0$, and it does not need to compute ψ_{ij}, φ_{ij}.

To remain the consistency of the secure policies, it is recommended to check the consistency of the rules set when a piece of rule has been cancelled. The triple concerning the rule cancelled is $V_i = (S, O, A)$, its decisions is $d = f(V_i) = f(S,0,A)$. After a piece of rule has been cancelled, the triple becomes $V_j = (S', O', A')$ with its $d' = f(V_j) = f(S', O', A')$. That is, $S \cap S' \neq \Phi$, $O \cap O' \neq \Phi$, $A \cap A' \neq \Phi$ and $(d, d') \in R$. Then the deletion of rules is not integrated. So all these entries which satisfy these relations should be treated. The rule concerning each new triple should be changed to be a new rule. The triple $V'' = (S'', O'', A'')$ of this new rule satisfies following relation:

$$S'' = S' - S\ ,\ O'' = O' - O\ ,\ A'' = A' - A\ .$$

From the description above, we know that the consistency checking can be executed in the system no matter when a piece of rule was added or cancelled in the ACL. So the ACL can remain its consistency.

In order to effectively perform the examination of the consistency of an ACL, further study should be undertaken in the future. Even though we have noticed that at least we can use two approaches to improve the examination. One is to remove the unnecessary computations using the information obtained from the sequential progress of the calculation of M_S, M_O, M_A, M_d. The other is to use a delta computation policy by making the system to keep a useful history record of the previous state of the consistency.

References

1. T. Ryutov and C. Neuman. Representation and Evaluation of Security Policies for Distributed System Services. In proceeding of the DARPA Information survivability conference Exposition, January 2000. Hilto Head, South Carolina.
2. R. S. Sandhu and P. Samarati. Access control: principles and practice. IEEE Computer, 40-48, Sept 1994.
3. S. M. Bellovin and W. R. Cheswick. Firewalls and Internet Security. Addison-Wesley Publishing Company, Inc., 1994.
4. HONG Fan and FU Xiao-qing, Discrete Mathematics. Huazhong University of Science &Technology publishing company, 1995.

Knowledge-Based Modeling and Simulation of Network Access Control Mechanisms Representing Security Policies

Jong-Young Koh, Mi-Ra Yi, Tae-Ho Cho, Hyung-Jong Kim, and Hong-Geun Kim

School of Electrical & Computer Engineering, Sungkyunkwan University,
Suwon, 440-746, South Korea
{jykoh, miracle,taecho}@ece.skku.ac.kr

Abstract. It is quite necessary that an organization's information network should be equipped with a proper security system based on its scale and importance. One of the effective methods is to use the simulation model for deciding which security policy and mechanism is appropriate for the complex network. Our goal is to build a foundation of knowledge-based modeling and simulation environment for the network security. Within this environment, users can construct the simulation model of security mechanisms, apply various security policies, and quantitatively analyze their security performance against possible attacks. In this study, we considered security domain and implemented the models based on a systematic modeling approach. We enabled the model to include knowledge in modular fashion and provided well-defined guidelines for transforming security policy to concrete rule set.

1 Introduction

Today, highly developed information and communication network is improving the efficiency of business and enhancing the quality of life. But on the other side, adverse effects such as illegal intrusion, effluence or compromise of information, or denial of service are increasing day by day. Therefore a security system that is suitable for the scale and the importance of an organization's network is quite essential [1]. Since evaluating the performance of a security system directly in real world requires heavy costs and efforts, an effective alternative solution is using the simulation model. In concrete terms, using the model we can build various simulation situations, perform iterative runs, and decide which security configuration is effective in meeting the change of network environment.

In this paper, we present a foundation of knowledge-based modeling and simulation framework for network access control, which has the following design goals. Be capable of modeling the security mechanism representing the policy as an adequately abstracted form. And be capable of simulating and analyzing the model with respect to security performance on altering the policy. To meet these goals, the scope of this study has the following phases.

In the phase of reviewing the network security, we will briefly mention security systems and selected a target to be modeled in this study, and investigate the selected target in regard to the structural and behavioral feature. Besides, we'll mention general security policies applicable to the system and investigate the existing well-defined methods for representing policies. In the phase of defining a modeling approach for the network security, we'll present a knowledge-based modeling and simulation framework facilitating the design and implementation of security models. Especially, we'll have the framework include the knowledge module based on the architecture of rule-based expert system so that security policies can be described as the form of rule sets. Moreover, we'll provide a systematic approach for transforming policies of highly abstracted level into rule sets of actually workable level in consideration of existing well-defined methods. In the phase of designing and implementing security models, we'll abstract meaningful characteristics of the target system from functional and dynamic point of view based on the defined modeling approach. And we'll test the validity of implemented models through the sample scenario of applying policy examples.

There are several existing papers similar and comparable to our paper. *Alain Mayer* designed and implemented a firewall analysis tool. This allows the administrator to easily discover and test the global firewall policy, uses a minimal description of the network topology, and directly parses the various vendor-specific low-level configuration files. Thus, the tool complements existing vulnerability analysis tools, as it can be used before a policy is actually deployed, and it operates on a more understandable level of abstraction [2]. *Noureldien* and *Osman* enumerate considerations that have to be taken into account in order to develop an appropriate and meaningful evaluation criteria, and propose a multi-dimensional criterion for evaluating firewalls. The criterion consists of three major components: security, performance and management. The analytical results of applying the proposed criteria on firewall show the strengths and benefits of the proposed multi-dimensional approach [3].

2 Review of Network Security

OSI security Architecture (X.800) classifies security services into several types such as authentication, access control, confidentiality, integrity, and non-repudiation. Security mechanisms can be classified into authentication, access control, encryption, and audit trail, and these require one or more algorithms or protocols [4]. As the core part of security infrastructure, access control mechanisms are widely used and can be implemented into various systems such as firewall, operating system, and database management system [4,5]. Therefore, we chose the packet filter, representatives of network level access control, as modeling target in this study.

2.1 Features of Access Control Mechanisms

In order to abstract main features to be represented in each model, we investigated the selected mechanisms in regard to the structural and behavioral feature.

Static Packet Filtering. Decides whether to pass or to drop each packet only by it's header fields. Packets are just examined by prearranged filtering rules regardless of the result of examining previous packets. Fig. 1 shows the behavioral feature of this mechanism. There are several advantages such as fast speed of processing, the easiness of collaborating with application level services, and the simplicity of implementation. But on the other side, there are serious disadvantages too. Though the attacker can fabricate the header of packet, it is impossible to examine the content of packet. And if an attacker accomplishes an intrusion, the harmful influence on the entire network is very critical [6].

Dynamic Packet Filtering (Stateful Inspection). Inspects the content of packet as well as the header, extracts required information in order to enforce the security policy, and maintains such information in the *Dynamic State Table*. Filtering is performed in consideration of the relationship of contiguous packets based on the table. Fig. 1 shows the behavioral feature of this mechanism [7].

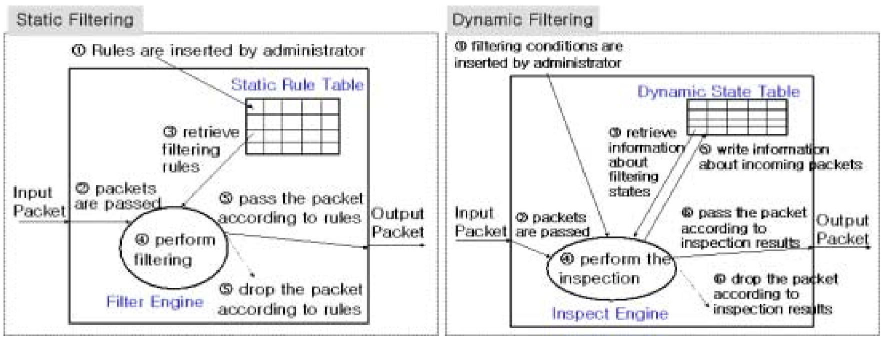

Static Filtering
① Rules are inserted by administrator
Static Rule Table
③ retrieve filtering rules
② packets are passed
Input Packet
④ perform filtering
⑤ pass the packet according to rules
Output Packet
⑤ drop the packet according to rules
Filter Engine
Dynamic Filtering
① filtering conditions are inserted by administrator
Dynamic State Table
③ retrieve information about filtering states
⑤ write information about incoming packets
Input Packet
② packets are passed
④ perform the inspection
⑥ pass the packet according to inspection results
Output Packet
⑥ drop the packet according to inspection results
Inspect Engine

Fig. 1. Process of the packet filtering

2.2 Representation of Security Policy

Setting-up the security policy is deriving concrete rules applicable to a certain system from abstract goals. It describes proper or improper activities towards protected resources based on security requirements that are confidentiality, integrity, and availability of resources [4,5]. Table 1 shows typical policy examples of detailed level. Those are concerned in major network access control issues common throughout diverse organizations and categorized according to applicable access control mechanisms [8].

There are several well-defined security policy models such as BLP (Bell-LaPadula), HRU (Harrison-Ruzzo-Ullman), Biba, and so on. Especially, the BLP is capable of representing the mandatory and the discretionary access control policy by way of more formalized method [9]. BLP is a state-machine model that represents the access permission through the access control matrix and the security level. The set of states is defined as $B \times M \times F$, where

$B = P(S \times O \times A)$ is the set of current accesses. An element $b \in B$ is a collection of tuples (s, o, a), indicating that subject s currently performs operation a on object o. S is a set of subject, O is a set of objects, and A is the set of access operations.

M is the set of access permission matrices $M = (M_{so})_{s \in S,\, o \in O}$.

F is the set of security level assignments. An element $f \in F$ is a tuple (f_S, f_C, f_O), where f_S gives the maximal security level each subject can have, f_C gives the current security level of each subject, and f_O gives the classification of all objects.

The HRU can be referred to represent policies that are related to modification of the access right and creation and deletion of the subject or the object, and also the Biba does for policies that are related to the integrity of access.

Table 1. Security policy examples

Applicable mechanism	Policies of detailed level
Static packet filtering	- Drop the packet spoofed with internal IP addresses - Pass the packet from trusted domains - Drop ICMP echo and direct broadcast packets - Drop the packet whose TTL value equal 1 - Shut down vulnerable services such as tftp, rlogin, etc.
Dynamic packet filtering	- Use the SYNdefender Relay method - Use the SYNdefender Gateway method - Use the Committed Access Rate function

3 Modeling Approach for Network Security

3.1 Knowledge-Based Modeling and Simulation Environment

In this paper, we propose the Structural Base and the Behavioral Base. The former is a conceptual framework for representing the structure of the complex network in clear and hierarchical manner based on the discrete event modeling theory, the latter is a code-level framework for representing the behavior of network components in modular and reusable manner based on the object-oriented paradigm.

Structural Base. We designed the Structural Base from the SES (System Entity Structure) that is a modeling methodology proposed by Bernard P. Zeigler, and it is capable of effectively representing the structural knowledge of the system [10]. Existing conceptual components of the system can be represented as entities that is to be models, and relationships among those entities can be represented as the following types:

- *Decomposition* (|): means that child entities are components of the parent.
- *Specialization* (||): means that child entities are kinds of the parent.
- *Multiple-Decomposition* (|||): means that multiple instances of child entities consist of the parent.

Fig. 2 shows an example of the Structural Base, which is a representation of network environment including the security system.

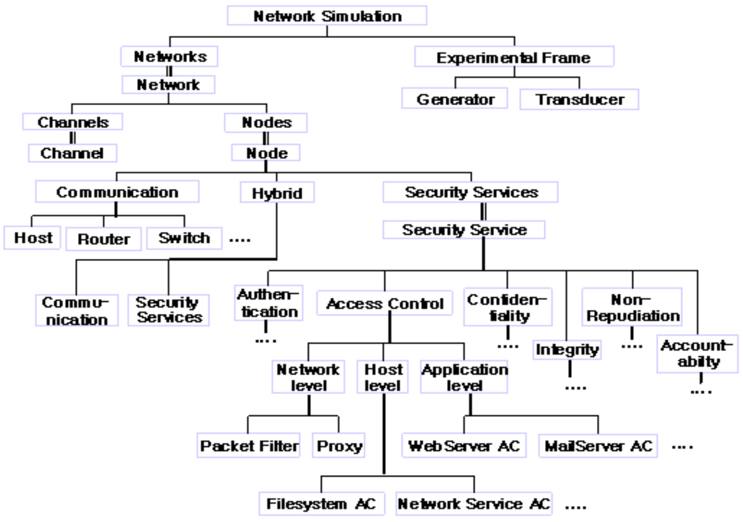


Fig. 2. An example of the Structural Base

Behavioral Base. We designed the Behavioral Base, which is composed of the Model Base, the Component Base, and the Rule Base like Fig. 3. These three types of 'Base' are capable of representing behavioral features of the system with different levels of abstraction and in modular fashion. The Model Base is a set of simulation models. The Component Base is a set of components of model. The Rule Base, as a particular component, is a set of rule subsets representing security policies.

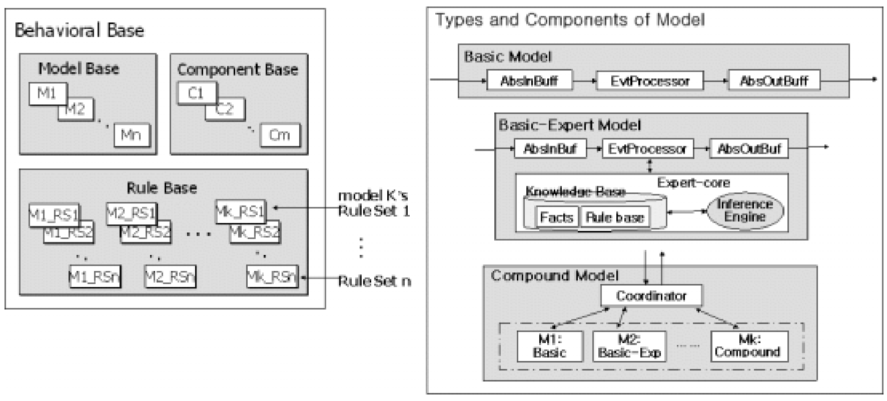


Fig. 3. The Behavioral Base, types, and components of model

3.2 Types and Components of Model

We designed two types of model based on the DEVS (Discrete Event System Specifications) proposed by Bernard P. Zeigler [10]. One is the Basic Model, and the other is the Compound Model.

Basic Model. As the model by which a unitary system functioning independently is represented, the Basic Model is defined by the structure $M = < X, S, Y, \delta, \lambda >$, where X is the set of external input event types, S is the sequential state set, Y is the set of external output event types, δ is the transition function dictating state transitions due to internal or external events $(S{\times}X{\rightarrow}S)$, and λ is the output function generating external events $(S{\rightarrow}Y)$.

Fig. 3 shows components of the Basic Model. The *EvtProcessor* facilitates handling of input events, state transitions of the model according to the passage of time, and generation of output events. The *AbsInBuff* (*AbsOutBuff*) facilitates buffering and conversion of input (output) data. As an extended type from the Basic Model, the Basic-Expert Model is capable of including a particular component named *Expert-Core*. The *ExpertCore* is based on the expert system architecture, and facilitates representation of security policies and decision-making.

Compound Model. The Compound Model groups one or more models so that these collaborate without conflicts. As the model by which more complex systems are represented, it is defined by the structure $CM = < \{M_i\}, \{I_i\}, \{Z_{ij}\} >$, where M_i is a component model, I_i is the set of influences on M_i, and Z_{ij} is a function translating output M_i to M_j. Fig. 3 shows components of the Compound Model. The Coordinator facilitates interaction among component models and interfacing with external.

3.3 Knowledge Representation inside Model

We present an approach to construction of knowledge base for the security model in consideration of formalized methods for representing policy (see Sect. 2.2) and general concept of the rule-based expert system, which facilitates representation of limited knowledge, accumulation of new knowledge, and separation of control [11].

To begin with, rule set mapping approach can be described as follows:

$$\begin{aligned}
&f : \overline{PA} \rightarrow R \quad \text{(P:Policy, A:Attack, R:Rule set)} \\
&\overline{PA} \sqsubseteq PA = P{\times}A = \{(p1, a1), (p2, a1), \ldots\} \\
&f = \{ ((p1, a1), r1), ((p2, a1), r2), \ldots \} \\
&P = \{ p1, p2, \ldots\},\ A = \{ a1, a2, \ldots\},\ R = \{ r1, r2, \ldots\}
\end{aligned} \tag{1}$$

And, several guidelines for constructing rule sets are defined.

- Security policies should be represented by the predicate logic.
- *Facts* should describe properties and relationships of the domain entity, and their evaluation result must be *true* or *false*.

- *Condition* part of the *IF* clause can have a series of *Facts* only if these are connected with the *AND* operator.
- For applying to the *Inference Engine*, one or more conflict resolution schemes should be considered.
- It is possible only to add a new rule, so modifying and deleting of existing one should not be happened.
- If rules are too increased, it is possible to separate rules into several groups according to steps of inference for decreasing conflicts and maintaining the consistency. Fig. 4 shows steps of inference. The process of inference is divided into subject discovering, object discovering, operation discovering and decision-making. Condition discovering can be included in any other steps.
- Subject, object, access operation, and access condition discovering rules are defined with respect to available information (initial facts) of security models.
- Access conditions are properties restricting particular subject, object, or operation. Security level and integrity level are dealt in access conditions too.
- All rules under the decision-making step should derive a goal state. The regular form of these rules by predicate logic is as the following.

"IF Who(Subject)$\wedge$What(Object)$\wedge$How(Operation) THEN Permit(Access)$\vee$Deny(Access)"

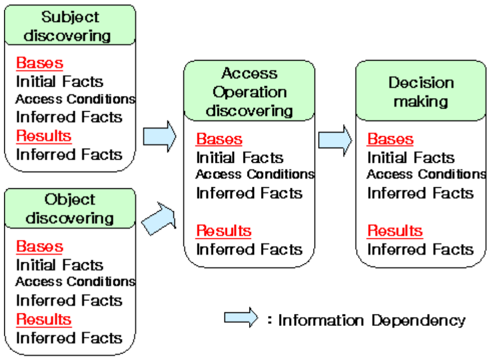


Fig. 4. Stepping the process of inference

4 Design of Security Model

4.1 Object Structure of Modeling Environment

Fig. 5 shows core objects provided by the modeling framework. These are designed based on the MODSIM III package. In the MODSIM III, a compile unit is an independent 'Module' file. The object structure of the framework is composed of 5 modules, 'Model-Type Module' defining model types, 'Security-Model Modules' defining specific security models, 'Component Module' defining components of the model, 'Entity Module' defining utility objects independently used in each model and 'Graphic-Resource Module' for GUI and animation.

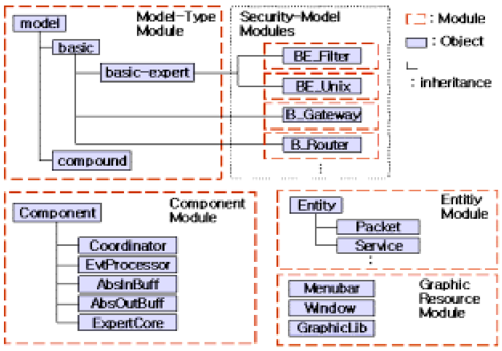


Fig. 5. Object structure of the modeling environment

4.2 Sample Scenario

Detailed applying scenario is necessary to evaluate security policy examples mentioned in Section 2.2 through the simulation, and then we constructed a sample scenario with regard to packet filtering policies and the SYN flooding attack. These are good representatives of the policy and the attack for the purpose of validating the proposed modeling framework.

Scenario 1. (allowance policy): TCP based services (such as Telnet, FTP, etc.) of internal network of an organization can be accessible from both internal and external network.

Scenario 2. (static packet filtering policy): As a result of scenario 1, the SYN flooding attack can be occurred from external attackers. As a countermeasure for this, packet filtering is performed using a designated list of deniable and permissible IP addresses. The organization precautions against the potentiality of attack by means of preventing all accesses from external to TCP services of protected hosts. But in this case, though all attack packets must be cut off clearly, normal accesses must also be restricted. So exceptionally, trustworthy hosts of external network are allowed to access those services.

Scenario 3. (dynamic packet filtering policy): As a result of scenario 2, the attack spoofed as a trustworthy host can still be occurred. As a countermeasure for this, consecutive SYN packets from the same client are dynamically monitored and prevented because those deserve to be an attack. For monitoring the state of packets dynamically, the packet filter can transmit the TCP connection-established packet instead of the target host (*SYNDefender Relay* [7]).

4.3 An Example of Security Model Design

Fig. 6 shows functional characteristics of the packet filter model designed based on the Basic-Expert Model. Inputs of this model are like policy scenario, filtering condition (allowable or preventable IP addresses, port number, TCP flag and so on), and continuously generated packets during simulation interval. Outputs of this model are permissible packets. Main processes inside this model are transition of model states, handling events due to components interaction, and inference engine deciding which packet is allowed.

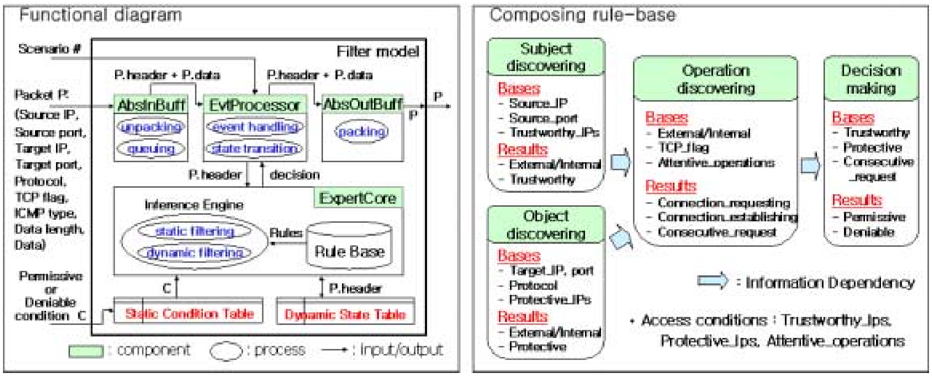


Fig. 6. Functional view and rule-base composition of the Filter model

Especially, as an instance of composition of rule-base in the model, Fig. 6 shows a process that compose the rule-base representing policies of the sample scenario in Section 4.2 according to the knowledge representation approach in Section 3.3. Table 2 shows the specification of completed rule-base. In the Fig. 10, available information of filter is major information extracted from arbitrary packets and additional information restricting specific subject, object and operation. The access condition of subject and object is a list of pre-inputted addresses. An access operation is determined to monitoring or not by TCP flag. Necessary objects and attributes to represent rules by predicate logic can be selected from available information of filter, and rules are grouped according to inference steps.

In the Table 2, FO_3 and FG_3 are rules for access control of protective objects, FS_3 and FG_4 are for trustworthy subjects, and from FP_1 to FP_5 are for attentive operations. To resolve the conflict, rules in upper side are applied prior to downward. Rules can be interpreted as the following examples. FS_3 means “If the source IP address is one of the trustworthy list, the subject is trusted”, FO_1 means “If the target IP address is internal, the port is FTP, and the protocol is TCP, the object is an internal FTP service”, and FP_1 means “If the subject is external, the protocol is TCP, and the TCP flag is SYN, the access operation is a request for connection”.

Table 2. The Specification of Filter model’s Rule-base

Steps	ID	Rule
Subject	FS_3	IF One_of(Source_ip, Trusty_ips) THEN Trusty(Subject)
Disco-	FS_1	IF External(Source_ip) THEN External(Subject)
Vering	FS_2	IF Internal(Source_ip) THEN Internal(Subject)

Table 2. (continued)

Steps	ID	Rule
Object Disco-vering	FO_3	IF One_of(Target_ip, Protective_ips) ∧ Tcp(Protocol) THEN Protective(Object)
	FO_1	IF Internal(Target_ip) ∧ Ftp(Target_port) ∧ Tcp(Protocol) THEN Internal_ftp(Object)
	FO_2	IF Internal(Target_ip) ∧ Telnet(Target_port) ∧ Tcp(Protocol) THEN Internal_telnet(Object)
Opera-tion Disco-vering	FP_1	IF External(Subject) ∧ Tcp(Protocol) ∧ Syn(Flag) THEN Connect_requesting(Operation)
	FP_2	IF External(Subject) ∧ Tcp(Protocol) ∧ Ack(Flag) THEN Connect_establishing(Operation)
	FP_3	IF Connect_requesting(Operation) ∧ ¬ One_of(Operation, Attentive_Ops) THEN append(Operation, Attentive_Ops)
	FP_4	IF Connect_establishing(Operation) ∧ One_of(Operation, Attentive_Ops) THEN remove(Operation, Attentive_Ops)
	FP_5	IF Connect_requesting(Operation) ∧ One_of(Operation, Attentive_Ops) THEN Consecutive_request(Operation)
Deci-sion Making (Goals)	FG_5	IF Consecutive_request(Operation) THEN Deniable(Access)
	FG_4	IF Trusty(Subject) ∧ Preventive(Object) THEN Permissive(Access)
	FG_3	IF External(Subject) ∧ Preventive(Object) THEN Deniable(Access)
	FG_1	IF External(Subject) ∧ Internal_ftp(Object) THEN Permissive(Access)
	FG_2	IF External(Subject) ∧ Internal_telnet(Object) THEN Permissive(Access)

5 Implementation of Security Model

Fig. 7 shows the execution window of simulation environment. During the execution, the window shows the movement of packets and the variation graph of performance index. Also, when clicking a model, statistical data until then can be shown. Provided functions are generating of security models, changing attributes of the model, compounding various models, representing security policies inside the model, simulating selected scenarios, animating the process of execution, displaying the summary of observation index and so on.

Before executing the simulation, a scenario must be selected in the menu and attributes of the model are initialized with appropriate values in the corresponding interface. Through the initialization interface for filter model, users can set up attributes of the filter model such as traffic limit, filtering way, deniable or permissible IP address, port number, TCP flag, and so on.

Each model represents final statistics when the simulation is finished. Table 3 is the result of sample scenario in the Section 4.2. We observed permission ratio of attack packets and denial ratio of normal packets in order to analyze security performance of the packet filter. Implemented model is valid because expected result is almost equal to simulation result. For example, when static packet filtering is applied to simulation, filter permits 60%, packet type 2, 5, 11, of total attack packets, that is, it denies 40% of attack packets.

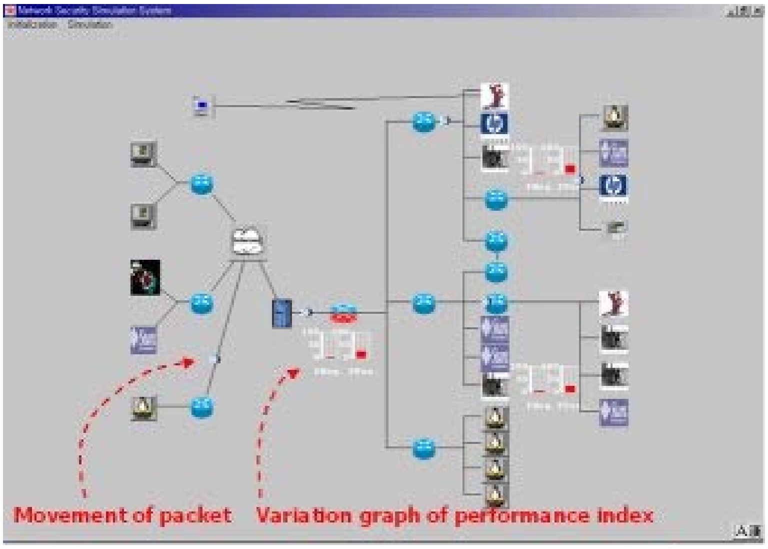

Network Security Simulation System
Initialization Simulation
Movement of packet
Variation graph of performance index

Fig. 7. Dynamic processing of simulation

Table 3. Simulation result of sample scenario

Packet Types	Source	target	flag	N/A	gener-ation (%)	allowance policy	Static filtering	dynamic filtering
1	Trusty	Protective	SYN	Normal	10	permit	Permit	permit
2			SYN	Attack	2.5	permit	permit	deny
3			ACK	Normal	12.5	permit	permit	permit
4		Accessible	SYN	Normal	10	permit	permit	permit
5			SYN	Attack	2.5	permit	permit	deny
6			ACK	Normal	12.5	permit	permit	permit
7	Untrusty	Protective	SYN	Normal	2.5	permit	deny	deny
8			SYN	Attack	10	permit	deny	deny
9			ACK	Normal	12.5	permit	deny	deny
10		Accessible	SYN	Normal	2.5	permit	permit	permit
11			SYN	Attack	10	permit	permit	deny
12			ACK	Normal	12.5	permit	permit	permit
Permission ratio of attack packets (%)						100	60	0
Denial ratio of normal packets (%)						0	20	20

6 Conclusion

In this paper, we presented a knowledge-based modeling and simulation environment for network access control, which is capable of modeling the security mechanism representing the policy as an adequately abstracted form and simulating and analyzing the model with respect to security performance on altering the policy. Through the presented environment, we can considerably save time required in evaluating the security system and expect the enhancement of security. From now on, we should make a step further from evaluating the role of security system itself. We will improve the environment to be capable of connecting the vulnerability database and analyzing the damage of resources with respect to more various scenarios. Thus, we will make the environment be a tool by which the security manager gets a comprehensive guidance.

References

1. F. Cohen : Simulating Cyber Attacks, Defences, and Consequences. Computer & Security. Vol.18 (1999)
2. A. Mayer, A. Wool, E. Ziskind : Fang: A Firewall Analysis Engine. Proceeding of IEEE Symposium on Security and Privacy (2000)
3. A. Noureldien, I.M. Osman : On Firewalls Evaluation Criteria. Proc. of TENCON 2000
4. An Introduction to Computer Security. NIST, Technology Administration, U.S. (1995)
5. C.M. King : Security Architecture. RSA press (2001)
6. E.D. Zwicky : Building Internet Firewalls. 2nd Ed. O'Relly & Associates (2000)
7. Avolio, Blask : Application Gateways and Stateful Inspection: A Brief Note Comparing and Contrasting. Trusted Information System, Inc. (1998)
8. Joel Scambry : HACKING EXPOSED 2nd Ed.: Network Security Secrets & Solutions. McGraw-Hill (2001)
9. Dieter Gollmann : Computer Security. Wiley (1999)
10. B.P. Zeigler *et al.*: Theory of Modeling and Simulation. 2nd Ed. Academic Press (2000)
11. K.W. Tracy : Object-Oriented Artificial Intelligence. Computer Science press (1997)
12. B.P. Zeigler, T.H. Cho, J.W. Rozenblit : A Knowledge-based Environment for Hierarchical Modeling of Flexible Manufacturing System. IEEE Trans. SMC A, Vol.26. (Jan. 1996)
13. T.H. Cho, H.J. Kim : DEVS Simulation of Distributed Intrusion Detection System. Transactions of the Society for Computer Simulation International, Vol.18. (Dec. 2001)

A Specification Language for Distributed Policy Control

Shigeta Kuninobu[1], Yoshiaki Takata[1], Daigo Taguchi[2], Masayuki Nakae[2], and Hiroyuki Seki[1]

[1] Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-1010, Japan
{shige-ku,y-takata,seki}@is.aist-nara.ac.jp
[2] Internet Systems Research Laboratories, NEC Corporation
8916-47 Takayama, Ikoma, Nara, 630-0101, Japan
{d-taguchi,m-nakae}@bp.jp.nec.com

Abstract. In this paper, we discuss a distributed policy control model where each object has its own policy and objects' behaviors are autonomously controlled based on those policies when they interact with one another. First the paper proposes a policy specification language suitable for distributed policy control. The operational semantics of the language is formally defined. Based on the formal semantics, we propose a runtime policy control mechanism for interpreting a given policy specification and sequencing method execution.

1 Introduction

Recent network technology enables us to exchange various kinds of data such as text, music and movie via a real-life computer network. Hence, a systematic method is needed to control the access to those data to achieve the required security and privacy of the system.

A *policy specification* is a set of rules (or *policies*), each of which describes when and in which condition a specified subject can (or cannot) perform a specified action on a specified target. An obligation policy such that s must perform a on t is important as well, especially in e-Business applications and network security management. In an object-oriented environment, an action corresponds to a method which may have arguments, and the above model is no longer useful to describe a policy.

A few policy specification languages are proposed which have rich functions to describe the above mentioned policies in a concise way. However, almost all the works implicitly assume the existence of a centralized policy controller. Also, defining a formal semantics of a policy specification language is important since a correct policy control is possible only by correctly interpreting the meaning of a given policy specification. However, how to define a formal semantics has been scarcely discussed except [7,2].

In this paper, we first propose a simple policy specification language, which is suitable for distributed policy control. The main features of the proposed specification language are summarized as follows.

- The language fits autonomous distributed policy control where an object has each own policy specification. In a specification, one can refer to a public attribute of another object as well as an attribute of the self object.
- A content-dependent conditional policy can be described concisely in the language.
- The language has a simple but useful exception handling mechanism by which policy violations and other exceptions are handled in a uniform way.
- The operational semantics of a policy specification is formally defined. This enables us to implement a reliable policy control mechanism which satisfies a given policy specification.

Related Works. Recently, several policy specification languages are proposed (for example, [3,4,5]). In [4] and its companion papers, logical languages based on Hoarn-clause are used as specification languages and various theoretical problems such as authorization inheritance and authorization conflict detection/resolution are discussed. However, these studies do not consider obligation policy. Ponder [3,2] is a general purpose policy specification language in which one can specify obligatory, conditional and data-dependent policies. Also a formal semantics of a subset of the language is defined in [7,2]. However, it is not easy to directly translate their semantics into a policy control mechanism. They define only the execution order of methods under a given policy specification, leaving the rest of the system as a blackbox. In contrast with [7], we describe the whole execution control of a system in a simple way by explicitly defining the change of runtime stack configuration. In [5], a specification language whose expressive power is no less than Ponder is proposed. The main advantage of the language is that both policy specifications and control targets can be maintained in a uniform XML environment. However, the formal semantics of the language is not provided.

2 Distributed Policy Control

This section shows how a policy for distributed objects can be specified. In a traditional access control model, 3-tuple (s, t, a) means that "subject s performs action a on target t" and is called an operation unit. In an object-oriented model, (s, t, a) corresponds to "subject s executes method a on target t." There are four kinds of basic access control policy for (s, t, a) as follows.

- positive authorization (or permission or right) : s is permitted to perform a on t.
- negative authorization (or prohibition) : s is forbidden to perform a on t by the target's policy.
- refrain : s is forbidden to perform a on t by the subject's policy.
- obligation : s is obliged to perform a on t (when a specific event has occurred).

We write $t.a \leftarrow s$ to denote operation unit (s, t, a). Furthermore, we write $t.a(p_1, \ldots, p_n) \leftarrow s$ to denote that s performs a with actual arguments $p_1, \ldots, p_n$ on t or s sends message $a(p_1, \ldots, p_n)$ to t.

In the following, auth+, auth−, oblg and refrain stand for positive authorization policy, negative authorization policy, obligation policy and refrain policy, respectively.

In our model, each object has its own policy. When more than one object interact with one another, the execution of every method should meet all the policies of the objects which participate in the method execution. For example, object A can play the movie contained in object B by executing method *B.play* only when both the policies of A and B permit A to execute *B.play*. We use the reserved word "this" to denote the self object, namely, the object which has that policy.

(1) positive authorization
```
    policy auth+ policy_name        this.m←B if Cond
```
"If *Cond* holds then subject B may perform action m on this object (or B may send message m to 'this' object)."

(2) negative authorization
```
    policy auth− policy_name        this.m←B if Cond
```
"If *Cond* holds then subject B must not perform m on this object."

(3) refrain
```
    policy refrain policy_name      B.m←this if Cond
```
"If *Cond* holds then this object must not perform m on target B."

(4) obligation
```
    policy oblg policy_name        B.m←this on Event if Cond
```
"If *Cond* holds when *Event* occurs then this object must perform m on target B."

Event should be a time instant (without duration). In the above policy specification,

- if *Event* ="beginning of $D.m' \leftarrow F$" then this object must perform m on B just before F performs m' on D.
- if *Event* ="end of $D.m' \leftarrow F$" then this object must perform m on B just after F performs m' on D.

2.1 Examples

Example 1 Policy of digital contents (Playing contents): Consider an object which has digital contents such as movie and music. This object may specify the following policy. Let x be an arbitrary user. "If x is the owner of this object and if x is older than or equal to 20 years then x may play the contents in this object. "

"Just after the execution of *play* method, x must execute *pay* method with actual arguments x, B and \$10.00. That is, when the contents have been played by x then x must pay \$10.00 to B."
```
policy auth+ Play_1
    var x: user
    this.play←x if this.owner==x, x.age>=20

policy oblg Play_2
    var x: user
    this.pay(x,B,$10.00) on end of this.play←x
```

Since this.pay(x,B,$10.00) which appears in Play_2 policy is an operation unit, it should be written as this.pay(x,B,$10.00)←this. If the subject of an operation unit is omitted in a policy, then the subject is specified as "←this" by default.

Example 2 Policy of a user (Refrain from playing contents) : A user object (or a personal computer of the user) may have the following policy. Let z be an arbitrary object which has a movie as contents.
"If the current user is under 18 years old and if the type of the contents is 'v' then this user cannot play z."

```
policy refrain Age_Check
    var z: content
    z.play if this.age<18, z.type==v
```

Instead of specifying Age_Check policy in the user object, you can put the following Age_Check_by_Contents policy in the movie object.

```
policy auth− Age_Check_by_Contents
    var x: user
    this.play←x if x.age<18, this.type==v
```

However, there is no guarantee that every movie object has such a policy as Age_Check_by_Contents. Even if the movie object has such a policy, it is difficult for a user to know it from outside. Therefore, it is desirable for both of a user object and a movie object to have policy specification and control mechanism.

3 A Policy Specification Language

In this section, we define the syntax of a policy specification language using BNF notation as follows. The policy specification language is a set of ⟨policy⟩.

```
⟨policy⟩ := 'policy' ⟨mode1⟩ ⟨policy name⟩
            [ 'var' ⟨variable declaration⟩+ ]
            ⟨operation unit1⟩+          [ 'if' ⟨condition⟩ ]
            % This rule defines the syntax of auth± and refrain policies.
          | 'policy' ⟨mode2⟩ ⟨policy name⟩
            [ 'var' ⟨variable declaration⟩+ ]
            ⟨operation unit2⟩+          'on' ⟨event⟩          [ 'if' ⟨condition⟩ ]
            % This rule defines the syntax of an obligation policy.
⟨condition⟩ := ⟨expression⟩     % The type of ⟨expression⟩ should be Boolean.
⟨mode1⟩ := 'auth+' | 'auth−' | 'refrain'
⟨mode2⟩ := 'oblg'
⟨operation unit1⟩ := ⟨object⟩'.'⟨action⟩ [ '←' ⟨object⟩ ]
⟨operation unit2⟩ := ⟨object⟩'.'⟨action⟩'(' ⟨expression⟩∗ ')'[ '←' ⟨object⟩ ]
% See table 1 for the microsyntax of ⟨operation unit1⟩ and ⟨operation unit2⟩.
⟨object⟩ := ⟨identifier⟩ | 'this'     % 'this' means the self object.
⟨expression⟩ := '(' ⟨expression⟩ ')' | ⟨object⟩'.'⟨attribute⟩ |
                ⟨expression⟩ ⟨binary operator⟩ ⟨expression⟩ |
                ⟨unary operator⟩ ⟨expression⟩ | ⟨constant⟩ | ⟨variable⟩
⟨event⟩ := 'beginning of' ⟨operation unit3⟩ | 'end of' ⟨operation unit3⟩
⟨operation unit3⟩ := ⟨operation unit1⟩
```

Table 1. Form of ⟨operation unit1⟩ and ⟨operation unit2⟩

	this.m←this	$x.m$←this	this.m←y	$x.m$←y
auth+	√		√	
auth−			√	
refrain	√	√		
oblg	√	√		

⟨variable declaration⟩ := ⟨variable⟩+ ':' ⟨type⟩
⟨policy name⟩,⟨action⟩,⟨attribute⟩,⟨variable⟩ := ⟨identifier⟩

- Note that, ⟨...⟩ are nonterminal symbols, A | B is a choice of A and B, [A] means A is an option, A∗ stands for 0 or more repetition of A, A+ stands for 1 or more repetition of A and 'α' stands for α itself as terminal symbols.
- The microsyntax of ⟨binary operator⟩, ⟨unary operator⟩, ⟨constant⟩, ⟨type⟩ and ⟨identifier⟩ are omitted.
- For ⟨operation unit1⟩ and ⟨operation unit2⟩, if ← ⟨object⟩ is omitted then ←this is specified by default.
- In table 1, x and y stand for any objects other than 'this.'
- As defined in the above BNF rules, ⟨operation unit3⟩ is used only in the event clause of obligation mode. It is allowed to have the form this.m ←this, $x.m$ ←this or this.m ←y where x and y stand for any objects other than 'this.'

4 Formal Semantics

In this section, we formally define a multi-object system in which the behavior of each object is controlled by specified policies. An object in the system calls a method of another object (or itself) along a given program, and the invocation of the called method is permitted or forbidden complying with both the caller's and callee's policies. Moreover, an invocation and an ending of a method may cause other obligatory method calls specified by the policies.

First, we define a simple model of objects and programs. Second, we introduce several concepts about policies and define the behavior of a system with policies. Finally, the system is extended by introducing an exception handling function. In our model, an exception occurs when a forbidden method call is requested, and thus policy violations are handled by the uniform exception handling function.

As a related work, Tonouchi and Damianou[2,7] define an operational semantics of policies written in Ponder. Their semantics is based on a system model which excludes the behavior of objects and a mechanism causing events which trigger off obligatory method calls. However, due to this simplification their semantics does not sufficiently describe the following features of the system.

- There is a recursive causal relation as: each policy does or does not become effective depending on the truth of its if-clause which may be affected by the state of the objects; the state of the objects is changed by the execution

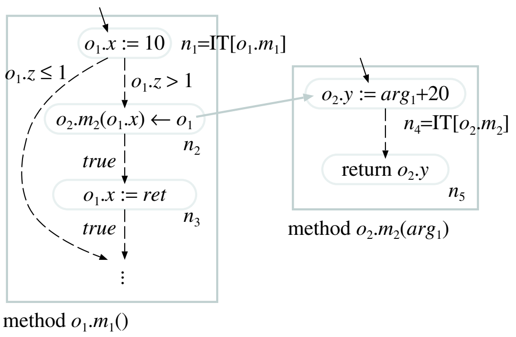


Fig. 1. A sample program

of a method; a method call is permitted or forbidden by a policy. Hence, to predict the correct behavior of the system, we have to consider the state of the objects and the state transition caused by the execution of a method.

– Executing an obligatory method call may cause other obligatory calls. To predict the chain of obligatory calls, we have to define the relationship between a method invocation and an event.

As described in the following subsections, our semantics is designed to include the above features into the formal semantics, though we currently omit some complex features such as multi-threaded execution.

4.1 Behavior of System without Policies

Objects. Each object has a finite number of attributes and methods defined by its class. Assume that an object o has attributes $a_1, a_2, \ldots, a_k$. When the value of a_i is v_i for $1 \leq i \leq k$, the state of o is represented by the tuple $state = (v_1, v_2, \ldots, v_k)$. We may write $o.a$ and $o.m$ to represent an attribute a and a method m of an object o, respectively.

Let O be a finite set of objects. Assuming a total order of objects in O, we let O be an ordered set $(o_1, o_2, \ldots, o_n)$. A global state of O is an n-tuple $\sigma = (state_1, state_2, \ldots,$
$state_n)$, where $state_j$ is a state of o_j for $1 \leq j \leq n$. Let $\sigma(e)$ denote the value of an expression e at a global state σ and let $\sigma[o.a := v]$ denote the global state which is the same as σ except that the value of the attribute $o.a$ is v.

Program. The body of a method $o.m$ is a program which is represented by a directed graph shown in figure 1. A program is a tuple (NO, TG, IS, IT, VAR). In the following we write NO[$o.m$], TG[$o.m$], and so on to represent each of the five components of the body of a method $o.m$. Let EXP[$o.m$] be the set of expressions each of which is a term consisting of built-in functions, attributes of o, and variables in VAR[$o.m$] (defined below).

- NO$[o.m]$ is a set of nodes which represent program points. We assume that for any o_1, o_2, m_1 and m_2, NO$[o_1.m_1]$ and NO$[o_2.m_2]$ are disjoint unless $o_1 = o_2$ and $m_1 = m_2$.
- TG$[o.m] \subseteq$ NO$[o.m] \times$ EXP$[o.m] \times$ NO$[o.m]$ is a set of edges called transfer edges. For any $n_1, n_2 \in$ NO$[o.m]$, $n_1 \overset{e}{\to} n_2$ denotes $(n_1, e, n_2) \in$ TG$[o.m]$, which represents that the control can move to n_2 just after the execution of n_1 if the value of e is *true*.
- IT$[o.m] \in$ NO$[o.m]$ is the entry point of the program and is called the initial node.
- IS$[o.m]$ is a mapping from a node to its label. The label of a node represents an atomic action and is one of the following forms.
  - $o_2.m_2(e_1, \dots, e_k) \leftarrow o$ Invoke $o_2.m_2$ with the arguments $e_1, \dots,$ $e_k \in$ EXP$[o.m]$: move the control to IT$[o_2.m_2]$ and assign the values of expressions $e_1, \dots, e_k$ to the parameters $arg_1, \dots, arg_k$ in VAR$[o_2.m_2]$, respectively.
  - return e Return to the caller method and move the control to the next node. The value of $e \in$ EXP$[o.m]$ is returned and is assigned to the special local variable *ret* of the caller method.
  - $o.a := e$ Assign the value of $e \in$ EXP$[o.m]$ to the attribute a of o itself.
  - $r := e$ Assign the value of $e \in$ EXP$[o.m]$ to the local variable $r \in$ VAR$[o.m]$.

  In the following, let IS be the mapping which is the union of IS$[o.m]$ for every method m of every object o.
- VAR$[o.m]$ is a set of local variables. Assuming that VAR$[o.m]$ is an ordered set $(r_1, \dots, r_k)$, a state of the local variables of $o.m$ is represented by a tuple $\mu = (v_1, \dots, v_k)$ of the values of them. We define $\mu(e)$ and $\mu[r := v]$ in the same way as the global state of objects. The value of an expression $e \in$ EXP$[o.m]$ at a global state σ and a state μ of local variables is $\sigma \circ \mu(e) = \sigma(\mu(e))$. The state of local variables in which the values of all variables are undefined is denoted by $\perp$.

4.2 Behavior of System with Policies

Policies. Now we extend the system described in the previous subsection by introducing policies. Consider an ordered set $O = (o_1, o_2, \dots, o_n)$ of objects. Each object in O has a finite set of policies as well as the attributes and the methods. Assuming that each object o_j has a set P_j of policies for $1 \leq j \leq n$, let $Policy(O) = \bigcup_{1 \leq j \leq n} P'_j$ where P'_j equals P_j except that "this" in P_j is replaced with o_j.

Let σ be a global state of the set O of objects, s and t be objects in O, m be a method of t, and *mod* be any of auth+, auth− and refrain. We define a relation $\sigma \models mod(s, t, m)$ as follows, which represents that the target t permits (if $mod =$ auth+) or forbids ($mod =$ auth−) the subject s to call the method m of t, or s refrains (if $mod =$ refrain) from calling m of t when the global state is σ.

$$\begin{aligned}\sigma \models \mathit{mod}(s,t,m) \quad \text{if and only if} \quad & \exists p \in \mathit{Policy}(O),\\ & \exists\theta\text{: a substitution for the variables in } p,\\ & p = \text{“policy } \mathit{mod} \ \ldots\ B.m \leftarrow A \text{ if } \mathit{Cond}\text{”},\\ & \sigma \circ \theta(\mathit{Cond}) = \mathit{true},\ \theta(A) = s,\ \theta(B) = t.\end{aligned}$$

If $\sigma \models \text{auth+}(s,t,m)$ and $\sigma \models \text{auth}-(s,t,m)$, then we say that σ causes a conflict between policies upon the operation (s,t,m). Using an arbitrary conflict resolution method (cf. [4,6]), we define $\text{CAN}(\sigma, t.m \leftarrow s)$ as a predicate which is true if the operation $t.m \leftarrow s$ is permitted when the global state is σ.

For a global state σ and an event ev, we also define the set $\text{oblg}(\sigma, ev)$ of obligatory method calls which become effective when the global state is σ and the event ev has just occurred.

$$\begin{aligned}&\text{oblg}(\sigma, ev) = \{\ t.m(v_1,\ldots,v_{k_m}) \leftarrow s \mid\\ &\quad \exists p \in \text{Policy}(O),\ \exists\theta\text{: a substitution for the variables in } p,\\ &\quad p = \text{“policy oblg} \ldots B.m(E_1,\ldots,E_{k_m}) \leftarrow A \text{ on } Ev \text{ if } \mathit{Cond}\text{”},\\ &\quad \sigma\circ\theta(Ev) = ev,\ \sigma\circ\theta(\mathit{Cond}) = \mathit{true},\ \theta(A) = s,\ \theta(B) = t,\\ &\quad \sigma\circ\theta(E_i) = v_i \text{ for } 1 \le i \le k_m\ \}\end{aligned}$$

Order of Obligations. If the set $\text{oblg}(\sigma, ev)$ of obligations has more than one elements, then we assume a total order $\prec$ over $\text{oblg}(\sigma, ev)$ which represents the order of performing the obligations. Thus

$$\text{oblg}(\sigma, ev) = \{op_1,\ldots,op_l\} \quad\text{and}\quad op_1 \prec op_2 \prec \cdots \prec op_l$$

where $op_i =$ “$t_i.m_i(v_{i1},\ldots,v_{ik_i}) \leftarrow s_i$” for $1 \le i \le l$. $op_i \prec op_j$ represents that op_i should be performed before op_j. For the above-mentioned $\text{oblg}(\sigma, ev)$, we define a sequence $\text{Foblg}(\sigma, ev)$ of stack frames as follows.

$$\text{Foblg}(\sigma, ev) = (n_{\text{oblg}}[op_1], \bot) : (n_{\text{oblg}}[op_2], \bot) : \cdots : (n_{\text{oblg}}[op_l], \bot),$$

where $n_{\text{oblg}}[op]$ is a special node newly introduced here for any obligatory operation op. Note that $n_{\text{oblg}}[op]$ does not belong to any method and $\text{IS}(n_{\text{oblg}}[op])$ is defined as $\text{IS}(n_{\text{oblg}}[op]) = op$. Let NO_{oblg} be the set of all $n_{\text{oblg}}[op]$s.

Transition System with Policies. The abstract multi-object system consists of a set O of objects and a control stack. The control stack is a sequence of an arbitrary number of stack frames. Each stack frame (or simply, frame) is a triple (n, μ, ef), where $n \in \text{NO}[o.m]$ for a method $o.m$, μ a state of the local variables of $o.m$, and ef a truth value. The frame represents that the control is at n in method $o.m$ with the state μ of the local variables, and if $ef = \mathit{true}$, then it also represents that the label of n is $o_2.m_2(e_1,\ldots,e_k) \leftarrow o$ and the control has already reached a return node in the callee method $o_2.m_2$. We may abbreviate a frame (n, μ, false) to (n, μ). The concatenation of two sequences ξ and ν is denoted by $\xi : \nu$. The empty sequence is denoted by ϵ. The rightmost symbol in a sequence of frames corresponds to the topmost symbol of the stack.

$$\text{(GASSIGN)}\quad \frac{\text{IS}(n) = \text{“}o.a := e\text{”} \quad n \overset{e_2}{\to} n_2 \quad \sigma' = \sigma[o.a := \sigma \circ \mu(e)] \quad \sigma' \circ \mu(e_2) = \mathit{true}}{(\sigma,\ \ \xi : (n, \mu)) \to (\sigma',\ \ \xi : (n_2, \mu))}$$

$$\text{(LASSIGN)}\quad \frac{\text{IS}(n) = \text{“}r := e\text{”} \quad n \overset{e_2}{\to} n_2 \quad \mu' = \mu[r := \sigma \circ \mu(e)] \quad \sigma \circ \mu'(e_2) = \mathit{true}}{(\sigma,\ \ \xi : (n, \mu)) \to (\sigma,\ \ \xi : (n_2, \mu'))}$$

$$\text{(CALL)}\quad \frac{\begin{array}{l}\text{IS}(n) = \text{“}o_2.m_2(e_1, \dots, e_k) \leftarrow o_1\text{”} \qquad \text{CAN}(\sigma, o_2.m_2 \leftarrow o_1) = \mathit{true} \\ f_2 = (\text{IT}[o_2.m_2], \bot[\mathit{arg}_1 := \sigma \circ \mu(e_1)] \dots [\mathit{arg}_k := \sigma \circ \mu(e_k)]) \\ \text{Foblg}(\sigma, \text{beginning of } o_2.m_2 \leftarrow o_1) = f_{\text{x},1} : f_{\text{x},2} : \cdots : f_{\text{x},l}\end{array}}{(\sigma,\ \ \xi : (n, \mu)) \to (\sigma,\ \ \xi : (n, \mu) : f_2 : f_{\text{x},l} : \cdots : f_{\text{x},2} : f_{\text{x},1})}$$

$$\text{(RETURN)}\quad \frac{\begin{array}{l}\text{IS}(m) = \text{“return } e\text{”} \qquad \text{IS}(n_1) = \text{“}o_2.m_2(e_1, \dots, e_k) \leftarrow o_1\text{”} \\ f_2 = (n_1, \mu[\mathit{ret} := \sigma \circ \mu_2(e)], \mathit{true}) \\ \text{Foblg}(\sigma, \text{end of } o_2.m_2 \leftarrow o_1) = f_{\text{x},1} : f_{\text{x},2} : \cdots : f_{\text{x},l}\end{array}}{(\sigma,\ \ \xi : (n_1, \mu) : (m, \mu_2)) \to (\sigma,\ \ \xi : f_2 : f_{\text{x},l} : \cdots : f_{\text{x},2} : f_{\text{x},1})}$$

$$\text{(FINISH)}\quad \frac{n_1 \notin \text{NO}_{\text{oblg}} \qquad n_1 \overset{e}{\to} n_2 \qquad \sigma \circ \mu(e) = \mathit{true}}{(\sigma,\ \ \xi : (n_1, \mu, \mathit{true})) \to (\sigma,\ \ \xi : (n_2, \mu))}$$

$$\text{(OFINISH)}\quad \frac{n_1 \in \text{NO}_{\text{oblg}}}{(\sigma,\ \ \xi : (n_1, \mu, \mathit{true})) \to (\sigma,\ \ \xi)}$$

Fig. 2. Inference rules which define the transition relation

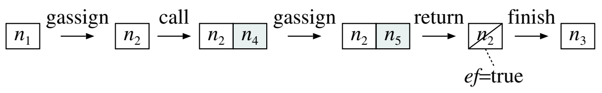


Fig. 3. Behavior of the system with the program in figure 1 and no obligation

The abstract system is represented by a transition system *Sys* defined as follows. A state of *Sys* is a pair (σ, ξ) where σ is a global state of O and ξ is the contents of the control stack. We define the transition relation $\to$ of *Sys* by inference rules in figure 2. Note that a method invocation $o_2.m_2 \leftarrow o_1$ is performed only when $\text{CAN}(\sigma, o_2.m_2 \leftarrow o_1) = \mathit{true}$. Moreover, when a method has been just invoked or has just finished, a sequence of stack frames which will accomplish the obligations caused by the beginning or the end of the method is added into the control stack (see figures 3 and 4).

4.3 Exception Handling

We extend our model by a function to handle exceptions, that is,

(1) Extend the program model so that we can specify an action to be performed when an exception has just occurred.
(2) Extend the transition system so that an exception occurs when a forbidden method call is requested.

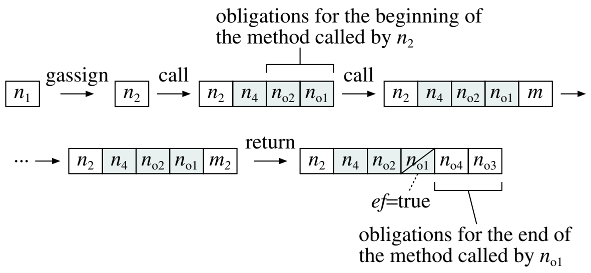

obligations for the beginning of the method called by n_2
gassign
call
call
return
ef=true
obligations for the end of the method called by n_{o1}
n_1
n_2
n_4
n_{o2}
n_{o1}
m
m_2
n_{o4}
n_{o3}

Fig. 4. Behavior of the system with obligations

```
      policy oblg DELIVERY_FEE
        var sender:Channel
        try
          (this.casher).pay((this.cert).fee, sender)
(a)     except
          on e:OperationFailed do this.throw(e)
        on beginning of this.receive←sender
```

Fig. 5. A sample obligation

Consider a situation in which an obligation causes a policy violation. In the definition of the policy specification language, we said that the main clause of an obligation policy is a method call, and thus we cannot specify any action for the violation exception. However, we extend the specification language as follows without changing the model of obligations. Figure 5 shows a specification of an obligation policy written in the extended language. In the language we can write an arbitrary program code in the main clause (part (a) of figure 5). We assume that the part (a) is the body of a method which has no name and when this obligation becomes effective, the method is called.

Program with Exception Handling. A program is a tuple (NO, TG, EG, IS, IT, VAR), where EG is a set of edges called exception edges. An element in EG is a tuple (n_1, r, ty, n_2) where $n_1, n_2 \in \text{NO}$, $r \in \text{VAR}$, and ty a type of an exception such as OperationFailed (see figure 6). Note that in our model an exception occurs only at a method call and thus n_1 should be a method call. $n_1 \overset{r:ty}{\rightarrow}_{\text{EG}} n_2$ denotes $(n_1, r, ty, n_2) \in \text{EG}$ and represents that if the control is at n_1 and an exception ex of the type ty occurs, it can move to n_2 assigning ex to r (i.e., the exception is caught). When an exception ex of a type ty occurs at a node n and $n \overset{r:ty}{\rightarrow}_{\text{EG}} n_2$ does not hold for any n_2 and r, ex is delivered to the method which called the method which n belongs to (i.e., the exception is thrown).

Transition System with Exception Handling. Let $\text{ex}_{\text{policy}}$ be a constant which represents the policy violation exception, and typeof(ex) be the type of an exception ex.

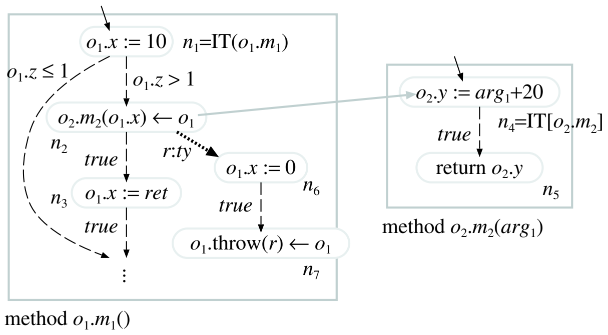


Fig. 6. A sample program with an exception edge

$$\text{(THROW1)}\quad \frac{\text{IS}(n) = \text{“}o_1.\text{throw}(e_1) \leftarrow o_1\text{”} \qquad ex_1 = \sigma \circ \mu(e_1)}{(\sigma,\ \ \xi : (n, \mu)) \rightarrow (\sigma,\ \ \xi : (n, \mu) : ex_1)}$$

$$\text{(POLICY\_EX)}\quad \frac{\text{IS}(n) = \text{“}o_2.m_2(e_1, \ldots, e_k) \leftarrow o_1\text{”} \qquad \text{CAN}(\sigma, o_2.m_2 \leftarrow o_1) = \mathit{false}}{(\sigma,\ \ \xi : (n, \mu)) \rightarrow (\sigma,\ \ \xi : (n, \mu) : \text{ex}_{\text{policy}})}$$

$$\text{(CATCH)}\quad \frac{n \overset{r:\text{typeof}(ex)}{\rightarrow_{\text{EG}}} n_2}{(\sigma,\ \ \xi : (n, \mu, ef) : ex) \rightarrow (\sigma,\ \ \xi : (n_2, \mu[r := ex]))}$$

$$\text{(THROW2)}\quad \frac{n \overset{r:\text{typeof}(ex)}{\rightarrow_{\text{EG}}} n_2 \ \text{ does not hold for any } n_2 \text{ and } r.}{(\sigma,\ \ \xi : (n, \mu, ef) : ex) \rightarrow (\sigma,\ \ \text{CASTOFF}(\xi : (n, \mu, ef)) : ex)}$$

Fig. 7. Inference rules for exception handling

Let $f_1 = (n_1, \mu_1, ef_1)$ and $f_2 = (n_2, \mu_2, ef_2)$ be arbitrary frames. We define a mapping CASTOFF from and to a sequence of frames as

$$\text{CASTOFF}(f_1) = \epsilon$$
$$\text{CASTOFF}(\xi : f_1 : f_2) = \begin{cases} \xi : f_1 & ef_1 = \mathit{true} \text{ or } n_2 \notin \text{NO}_{\text{oblg}} \\ \text{CASTOFF}(\xi : f_1) & \text{otherwise.} \end{cases}$$

We extend the transition system *Sys* as follows. A stack frame can be either the above-mentioned tuple (n, μ, ef) or an exception ex. If the topmost element of the control stack is an exception ex, it represents that ex has occurred and is to be processed. We add the inference rules in figure 7 to the set of the rules in subsection 4.2. We also add “$m_2 \not\models$ throw” to the premise of the rule (CALL) in figure 2.

5 Implementation Issues

In this section we describe a policy control system, which is an implementation of the abstract multi-object system defined in the previous section, using an

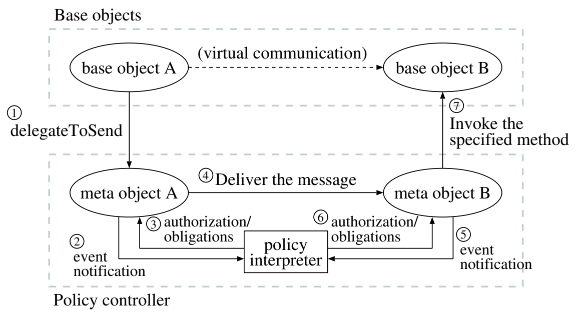


Fig. 8. Communication using meta objects

```
class ObjectA extends BaseObject {
  void methodA() {
    BaseObject objB = PolicyController.getObject("somewhere/objectB");
    Message msg = new Message("methodB", new Object[] {a1, a2});
    sendMessage(msg, objB); // Send a message toward objectB.
} }
class ObjectB extends BaseObject { void methodB(Class1 p1, Class2 p2) { ... } }
```

Fig. 9. An implementation of base objects

object-oriented programming environment such as Java. We call a member of the concerned multi-object system a base object. To each base object, we attach a newly introduced object called a meta object in the same way as the reflective authorization system described in [1]. Each base object cannot communicate directly with another one. Instead, it delegates its meta object to send or receive a message to or from another (see figure 8). Figure 9 is an implementation of base objects in Java, where methodA of a base object of class ObjectA calls methodB of another base object of class ObjectB. Note that methodA does not call methodB directly but uses sendMessage of the superclass BaseObject instead. sendMessage invokes delegateToSend of the meta object of the caller. The meta object delivers the message to the callee's meta object, and then receiveMessage of BaseObject is invoked. Finally receiveMessage invokes the specified method, that is, methodB. The policy control is achieved by the meta objects and a policy interpreter. At first the meta object of caller object checks whether the method call is permitted or not. The policy interpreter computes the truth of the predicate CAN in subsection 4.2 and if it is false, throws the policy violation exception. Next the meta object performs the operation as well as the obligations.

6 Conclusion

In this paper, a policy specification language suitable for distributed policy control is proposed and its formal semantics is provided. There are several theoretical issues to pursue, namely, policy conflict, termination and confluence [4,6].

Acknowledgments

The authors would like to thank Prof. Y. Kaji of Nara Institute of Science and Technology, Mr. T. Fujita, Mr. H. Shimazu, Mr. T. Ichiyama, Mr. R. Ogawa and Mr. I. Hosomi of NEC corporation for their supports and valuable discussions.

References

1. M. Ancona, W. Cazzola and E. B. Fernandez: Reflective authorization systems: Possibilities, benefits, and drawbacks, *Secure Internet Programming*, LNCS 1603, 35–49, 1999.
2. N. C. Damianou: *A Policy Framework for Management of Distributed Systems*, PhD thesis, Imperial College of Science, Technology and Medicine, 2002. http://www-dse.doc.ic.ac.uk/Research/policies/ponder/thesis-ncd.pdf.
3. N. C. Damianou, N. Dulay E. Lupu and M. Sloman: The Ponder policy specification language, POLICY 2001, LNCS 1995, 18–38.
4. S. Jajodia, P. Samarati and V. S. Subrahmanian: A logical language for expressing authorizations, 1997 IEEE Symp. on Security and Privacy, 31–42.
5. M. Kudo and S. Hada: XML document security based on provisional authorization, 7th ACM Conf. on Computer and Communication Security, 87–96, 2000.
6. E. C. Lupu and M. Sloman: Conflicts in policy-based distributed systems management, IEEE Trans. on Software Engineering, 25(6), 852–869, 1999.
7. T. Tonouchi: *An Operational Semantics of a 'Small' Ponder*, July 2001. http://www.doc.ic.ac.uk/˜tton/Semantics.pdf.

Access Control Infrastructure for Digital Objects[*]

Javier López, Antonio Maña, Ernesto Pimentel, José M. Troya, Mariemma I. Yagüe

Computer Science Department
University of Málaga
{jlm,amg,ernesto,troya,yague}@lcc.uma.es

Abstract. Distributed systems usually contain objects with heterogeneous security requirements that pose important challenges on the underlying security mechanisms and especially in access control systems. Access control in distributed systems often relies on centralized security administration. Existing solutions for distributed access control do not provide the flexibility and manageability required. This paper presents the XML-based Secure Content Distribution (XSCD) infrastructure is based on the production of self-protected software objects that convey contents (software or data) and can be distributed without further security measures because they embed the access control enforcement mechanism. It also provides means for integrating Privilege Management Infrastructures (PMIs). Semantic information is used in the dynamic instantiation and semantic validation of policies. XSCD is scalable, facilitates the administration of the access control system, guarantees the secure distribution of the contents, enables semantic integration and interoperability of heterogeneous sources, solves the “originator retained control” issue and allows activities (such as payment) to be bound to the access to objects.

Keywords: Distributed systems security, secure content distribution, XML metadata, Privilege Management Infrastructure.

1 Introduction

The “digital object” concept has proven to be a valuable approach for different applications in distributed environments. Digital objects can be classified with respect to their contents. On the one hand, data objects encapsulate several logically related pieces of data along with some administrative information in a package intended to provide a uniform access. This is the case in systems for information commerce, digital libraries or eBooks. On the other hand, software objects encapsulate several services/operations. Software objects are found in object oriented middleware, web services or grid computing.

The security problems associated to digital objects also depend on the type of content encapsulated in the object. The protection for data objects is usually related to *Digital Rights Management* issues. In particular, access control, use-control, payment and copyright enforcement are relevant problems. The main problem for software

[*] Work partially supported by Spanish *Ministerio de Ciencia y Tecnología* under project TIC2002-04500-C02-02

objects is use-control, while access control (to operations), copy-protection, function analysis and runtime protection are also important issues. In both cases, the ability to retain control over the objects after they are accessed, known as "originator retained control", is a desirable security property.

Distributed systems usually contain objects with heterogeneous security requirements. However, some of the new scenarios where distributed systems are emerging share some common problems. The most remarkable ones are the following. Firstly, it is usual that objects are accessed by previously unknown users. Therefore, subscription-based schemes are not appropriate in this case. Secondly, the execution of copyright agreements, payment or other activities must be bound to the access to the objects. Finally, the originator or owner of the object must retain control over it regardless of its physical location and even after it is accessed by users. Other requirements are: (i) that a high degree of flexibility is required because of the heterogeneous nature of the objects, (ii) that being able to change the access control parameters dynamically and transparently is also essential and, (iii) due to the large amount of objects, it is important to be able to establish access conditions in an automatic way based on information about objects.

Paradoxically, access control in distributed systems often relies on centralized security administration. Centralized control has important disadvantages: (a) the control point represents a weak spot for security attacks and fault tolerance, (b) it does not facilitate the deployment of owner retained control mechanisms, (c) it reduces system performance because it introduces a bottleneck for request handling, and (d) it usually enforces homogeneous access control schemes that do not fit naturally in heterogeneous user groups and organizations. On the other hand, systems for distributed security administration still have open problems. Solutions proposed so far do not provide the flexibility and manageability required. A system for distributed access control has been proposed based on the concept of mobile policies [1] to solve some of the limitations of *Role Based Access Control* (RBAC) schemes [2]. This improvement is limited by the requirement of executing the access control policies in trusted computers. Furthermore, when access to an object is granted, this object is sent to the client computer where it has no protection. Finally, because object and policy are compiled in a package, a change in the policy requires that the object-policy package is recompiled and distributed to all trusted servers.

This paper presents the *XML-based Secure Content Distribution* (XSCD, pronounced "exceed") infrastructure that provides distributed access control and enforcement and secure content distribution. We address the integration of a separate *Privilege Management Infrastructure* (PMI) by defining mechanisms for the semantic description of its components. We introduce *Semantic Policy Language* (SPL), an XML-based policy definition language designed to specify policies in a simple way, to be evaluated by processors with limited capabilities such as smart cards and to facilitate semantic policy validation processes. SPL policies are modular and can be composed without ambiguity. We also address the problem of the association of policies to objects in a flexible and automated way and the combination of policies. To achieve our goals we have extended the concept of mobile policy by allowing their execution in untrusted systems and used XML metadata technologies extensively.

The rest of the paper is organized as follows. Section 2 summarizes some relevant related work. Section 3 presents the building blocks of XSCD. Section 4 describes the infrastructure. Finally, section 5 summarizes the conclusions and presents ongoing and future work.

2 Related Work

Regarding access control, several proposals have been introduced for distributed heterogeneous resources from multiple sources [3][4]. Unfortunately, these proposals do not address the specific problems of access control in distributed systems. Traditional access control schemes such as mandatory access control (MAC), discretionary access control (DAC) or RBAC are not practical for scenarios where the users are previously unknown or with a very large number of registered users.

Static grouping of users can suffice in many situations but it is not flexible enough to cope with the requirements of more dynamic systems where the structure of groups can not be anticipated by the security administrators. In these scenarios new resources are frequently incorporated to the system and each resource may need a different group structure and access control policy. Furthermore, the policy for a given resource may change frequently. A different approach is required in order to solve the scalability problems of these systems, facilitate access control management and provide means to express access conditions in a natural and flexible way.

Some systems based on the idea of the self-secured package of information have been proposed for secure content distribution. None of these systems has achieved a representative use because the security of these systems depends heavily on the security of the client software. IBM's *Cryptolope* [5] is one of the most elaborated alternatives. A Cryptolope is a package that includes the protected content and all necessary administrative information. As noted in [6], the fact that the opener component (known as Cryptolope Player) runs on the end user's PC, introduces the possibility to produce software emulators. In addition, an infrastructure of *trusted clearing houses* and online connection with these entities is needed. A similar scheme is Intertrust's *Rights|System platform* [7] where digital content is protected even when it is resold. This platform is designed for high-value digital goods but is actually limited to three data formats. Both schemes are platform dependent (need specific client software), offer a set of closed possibilities for the contents, and have no integrated payment scheme. Moreover, they are designed for high-value digital goods, but are not adequate for low-value transactions and occasional business relations.

In the context of policy specification, several XML based languages such as XACL, XrML, ODRL, ebXML, SAML or XACML have been developed for access control, digital rights management, authentication and authorization. Many similarities and interesting features can be found among these languages. Nevertheless, they do not support some relevant properties such as policy parameterisation and composition. Moreover, many features provided by those languages are not necessary in our application scenarios [8].

Two related proposals are the Author-X system [9] and the FASTER project [10], which propose similar systems specific for access control to XML documents. Both systems define hierarchic access control schemes based on the structure of the document. But the structuring of XML documents does not necessarily match the security requirements of the nodes. As a consequence, for the general case the number of different authorizations (positive and negative) that have to be defined grows up rapidly. Author–X policy language uses DTDs, while FASTER uses XML-Schema [11]. Scalability is very limited in both Author-X and FASTER systems. The FASTER system is described as completely server-side and Author-X is essentially centralized, although a distributed approach is proposed based on a set of XML federated sources relying on a central 'master source'. The design based on this central 'master source' has negative consequences on its scalability.

The content protection of Author-X is founded on the concept of "passive" secure container requiring a different key for each possible view of the document. This introduces important disadvantages related to the administration of the access control system and the security [12]. FASTER does not support any content protection mechanism, except the creation of the appropriate user view on the server. FASTER access control is based on user groups and physical locations defining a subject hierarchy. This scheme does not work well for scenarios where heterogeneous contents are frequent and the structure of groups can not be anticipated by the administrators. Author-X is based on credentials that are issued by the access control administrator. Therefore, in practice, each credential will be useful only for a single source, limiting interoperability. A consequence of this approach is that users are obliged to subscribe to sources before they can access their contents

3 XSCD Overview

The main motivation has been to design an access control scheme that is scalable to a large number of previously unknown users and solves the originator retained control issue. Starting from a design based on attribute certificates and taking into account the characterization of different scenarios as well as the analysis of previous proposals. The following are the main goals for our secure distribution infrastructure:

- *Scalability.* The centralized approach adopted in current access control systems introduces many drawbacks in terms of efficiency, manageability and security. The access control system must be designed to suit scenarios where the number of users, attributes and policies are very large. Therefore, distributed access control enforcement is essential. Moreover, distributed management of the security policies is also important.
- *Originator-retained-control.* This issue deals with enabling the originator to retain control over the protected object, not over the contents. The latter, called 'full use-control', is a digital rights issue that is out of the scope of this work.
- *Distributed access control management.* Administrators should be able to manage their resources regardless of the resource location. Attribute certificates issuers must be independent of the application and the system must support the secure interoperation with those entities.
- *Interoperability.* The integration of different heterogeneous object sources is hindered by traditional access control systems because each source defines a specific access control scheme. The integration of an external PMI represents a step towards the solution of this problem.
- *Distributed access control enforcement.* Access control enforcement mechanisms must be distributed in order to avoid bottlenecks in request processing.
- *Ease of management.* The distributed approach must not introduce complexity of management. Tools to help security administrators should be provided.

The separation of the access control and authorization functions (credential issuance or attribute certification in our case) is universally accepted as a secure and scalable approach. XSCD is based on the integration of an external PMI supported by semantic information about the certification entities. We describe now the basic building blocks of the XSCD infrastructure.

3.1 Building Blocks

Privilege Management Infrastructure. Traditionally, the study of authorization issues has focused on access control. However, when considering Internet applications, authorization adopts a wider meaning, including group membership, role identification (collection of permissions or access rights, and aliases for the user's identity), limits on the value of transactions, access time for operations, security clearances, time limits, etc. Attribute certificates provide, for those applications, the means to carry authorization information, which in this way becomes "mobile".

The mobility feature of attributes is not a new issue. In fact, extensions of identity certificates as specified in ITU-T 1997 recommendation [13] tried to address this problem. However the use of the corresponding extension, *subjectDirectoryAttributes*, does not make entity attributes independent from identity. To be more precise, when using that solution, the change of privileges indirectly force a costly revocation of the identity related information. Besides, that solution does not solve delegation and impersonation issues, which are especially relevant in many of actual applications. The ITU-T 2000 recommendation [14] provides a more suitable solution because it clearly defines a framework where identity and attribute certificates, although related, can be independently managed. That recommendation defines new types of authorities, *Attribute Authorities* (AA), for the assignment of privileges. It also defines the *Source of Authority* (SOA) as the ultimate authority to assign a set of privileges. Additionally, the ITU-T framework provides a foundation to build a PMI that contain a multiplicity of AAs, SOAs and final users.

Usually, each SOA issues certificates for a small number of semantically related attributes. With this approach security administrators do not have control over some elements of the access control system. Consequently, a mechanism to establish the trust between these administrators and the PMI is required. We have addressed this problem using semantic information about the certifications issued by each SOA to assist the security administrators in the creation and semantic validation of access control policies.

Software Protection. In order to provide secure content distribution and originator control, the access control system must provide means to protect the contents not only while in transit through the network but also when they arrive to the destination. The problem with systems based on passive secure information containers is that users must install specific client software to access the protected data. This software controls access to the data and enforces the appropriate actions (e.g. payment) before access is granted. Because the client software is not protected, it becomes the weak spot in terms of security. Opposed to these proposals we use "active" containers (software instead of data) in order to avoid some problems of the latter. XSCD uses protected mobile software elements named *Protected Content Objects* (PCO) to convey the contents and force the user to fulfil the applicable policy before access is granted. By "protected software" we mean that it is neither possible to discover nor to alter the function that the software performs and it is also impossible to impersonate the software. In our solution, this is achieved using a variant of the SmartProt system

[15]. SmartProt partitions the software into functions that are executed by two collaborating processors. One of those processors must be a trusted computing device that enforces the correct execution of the functions and avoids that these functions are identified or reverse engineered. We are currently using smart cards for this purpose although other alternatives are possible.

Some specific sections of the unprotected software are translated by SmartProt into functionally equivalent sections of card-specific code. The translation process also identifies the dependencies between these protected sections, reorganizes the code and introduces fake code and data to confuse the attacker. These sections are then encrypted with a unique, randomly produced key using a symmetric cryptosystem. This key will be later included in a license (for a specific smart card) that is required to be able to run the software. Each license is encrypted using the public key of the client smart card. The last step substitutes the original code sections by calls to a function that transmits the respective equivalent protected sections, including code and data, to the card. Some additional support functions are also included. Therefore, the protected sections of the software do not reside in the cards; instead, during the execution of the software, these sections are transmitted dynamically as necessary to the card where they are decrypted using the installed license and executed. When finished, the card may send back some results. Some other partial results will be kept in the card in order to obtain a better protection against function analysis and other attacks. As each piece of software has its own key, we can manage them individually, which is not possible in other software protection proposals where the protected sections of all applications share the same key. For XSCD we integrate the access control policy and the license in a structure that we call Mobile Policy (MP).

Metadata. Most of times, metadata (information about data) are designed to support people or programs in locating and retrieving information resources. XML metadata technologies such as XML-Schema, RDF and RDF Schema [16] provide the foundation for the description of semantic information in our proposal. Metadata are applied at different levels in XSCD. On one hand, access control policies benefit from metadata for its creation and semantic and contextual validation. Likewise, digital objects have metadata associated that are used for the dynamic policy assignment and parameter instantiation. Additionally, metadata are used at the mobile policies creation level, for the specification and acquisition of certification rules. On the other hand, metadata is an essential tool for the integration of the external PMI.

Authorization language. Although other XML-based languages have been developed for access control and authorization, there are specific requirements found in our infrastructure that make their use difficult or inadequate. The main reason is that they do not support some relevant properties such as policy parameterisation and composition. On the other hand, policies must be processed inside smart cards with limited storage and processing capabilities. Therefore, the specification language must be simple enough to be translated into a more compact form in order to be processed by the smart cards. For these reasons, we have developed a specific XML-Schema based language, *Semantic Policy Language* (SPL), in order to specify the access control policies. The keys to the high flexibility of SPL are the extensive use of metadata, the modular composition of policies that separates declaration of each policy component and the parameterisation of the policies.

The SPL system uses several components to define policies. *SPL Policies* specify the conditions that must be satisfied to gain access. Each *Policy Applicability Specification* (*PAS*) links some policies to a series of resources. Finally, *Secured Resource Representations* (*SRRs*) and other contextual metadata are used to provide semantic information that is used in policies and PAS. Additionally, *Source of Authorization Descriptions* (*SOADs*) provide a semantic description of the PMI that is essential for the policy validation process. Fig. 1 shows examples of some of these components.

SPL policies are described following an XML-Schema template where we can declare access rules stating the set of certificates that must be presented for granting access. Optionally, we can declare a set of actions to be performed before access is granted. Examples of these actions are `Notify_To`, `Payment` and `Online_Permission`. This is known as provisional authorization. *Import* clauses can also be included to substitute some of the previous components of the policy and to allow the modular composition of policies based on the X-Path standard. Additionally, dynamic instantiation is enabled by the possibility to define parameters in the policies. The instantiation of parameter references is stated in the PAS. Metadata (SRRs and contextual information) is used for parameter instantiation.

The *Policy Applicability Specification* (*PAS*) provides an expressive way to relate policies to resources, either explicitly or based on metadata. PAS documents include declarations of the applicable policies, the target objects and, optionally the instantiation of the parameters of the policy. The object declaration includes the object location, the operations affected (defaulting to all operations) and some optional conditions. In this way, the PAS comprise all necessary information to relate policies to objects, and to instantiate policies. The PAS in Fig. 1 relates all objects in 'http://www.lcc.uma.es/Research/VFwkProgramme' of type 'report' to the policy defined in 'VFrameworkProgram.xml'.

The *Secured Resource Representation* (*SRR*) is a simple and powerful mechanism to describe properties about resources. Properties described in SRRs are used to instantiate policies and PAS, and to locate the applicable policies. The SRR in Fig. 1 declares that object 'http://www.lcc.uma.es/Research/VfwkProgramme/WP3.pdf' is a 'report' and belongs to the project with ID 'IST_2001-32446'.

The *Source Of Authorization Description* (*SOAD*) documents are digitally signed [17] RDF instances expressing the different attributes certified by each SOA, including their names, descriptions and relations. SOADs convey information that is essential for the semantic validation of the policies such as metadata about the different attributes certified by the SOA and its certification procedures. The set of SOADs represents the semantic description of the PMI. Full integration of the PMI can be achieved transparently for the rest of the system based on this description.

4 Architecture and Operation of the Infrastructure

Fig. 2 shows the main components of the XSCD system and their relationship. The first component is called *Policy Assistant*. This component uses the SOADs to produce and validate SPL Policies and PAS. The second component is the *SmartProt* protection system. This component transforms unprotected content objects in the server into *PCOs* generating also their corresponding *Licenses*. A PCO is a software object that encapsulates and protects the original object and enforces the access control mechanism. PCOs can be freely distributed to untrusted servers. It can be noticed that policies are not included in the PCO. The third component, called *Mobile Policy Generator*, attends requests from end users producing *MP*s dynamically. The *Object Metadata* database, containing SRRs, is used to determine the set of applicable policies for the corresponding PCO.

```
<?xml version="1.0" encoding="UTF-8"?>
<spl:policy xmlns:spl="http://www.uma.es/ICICS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.uma.es/ICICS Policy.xsd">
  <spl:parameter>Partner</spl:parameter>
  <spl:parameter>Project</spl:parameter>
  <spl:access_Rules>
    <spl:access_Rule>
      <spl:attribute_Set>
        <spl:attribute>
          <spl:attribute_Name>Member</spl:attribute_Name>
          <spl:attribute_Value>*Partner[@name]</spl:attribute_Value>
          <spl:SOA_ID>*Partner[@SOA]</spl:SOA_ID>
        </spl:attribute>
        <spl:attribute>
          <spl:attribute_Name>Project</spl:attribute_Name>
          <spl:attribute_Value>*Project</spl:attribute_Value>
          <spl:SOA_ID>*Partner[@SOA]</spl:SOA_ID>
        </spl:attribute>
      </spl:attribute_Set>
      <spl:attribute_Set>
        <spl:attribute>
          <spl:attribute_Name>Role</spl:attribute_Name>
          <spl:attribute_Value>Commisary</spl:attribute_Value>
          <spl:SOA_ID>EU_SOA</spl:SOA_ID>
        </spl:attribute>
        <spl:attribute>
          <spl:attribute_Name>Supervisor_Of</spl:attribute_Name>
          <spl:attribute_Value>*Project</spl:attribute_Value>
          <spl:SOA_ID>EU_SOA</spl:SOA_ID>
        </spl:attribute>
      </spl:attribute_Set>
      <spl:attribute_Set>
        <spl:attribute>
          <spl:attribute_Name>External_Reviewer</spl:attribute_Name>
          <spl:attribute_Value>*Project</spl:attribute_Value>
          <spl:SOA_ID>EU_SOA</spl:SOA_ID>
        </spl:attribute>
      </spl:attribute_Set>
    </spl:access_Rule>
  </spl:access_Rules>
</spl:policy>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<spl:PAS xmlns:spl="http://www.uma.es/ICICS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.uma.es/ICICS pas.xsd">
<spl:policy>VFrameworkProgram.xml</spl:policy>
  <spl:object>
    http://www.lcc.uma.es/Research/VfwkProgramme
  </spl:object>
  <conditions>
    <condition>
      <property_Name>object_Type</property_Name>
      <predicate>equals</predicate>
      <property_Value>report</property_Value>
    </condition>
  </conditions>
  <spl:instantation>
    <spl:formal_Parameter>Project</spl:formal_Parameter>
    <spl:actual_Parameter path="Project_ID">
      <!—empty because is instantiated from the SRR -->
    </spl:actual_Parameter>
  </spl:instantation>
  <spl:instantation>
    <spl:formal_Parameter>Partner</spl:formal_Parameter>
    <spl:actual_Parameter path="//Members/Partner">
 http://www.lcc.uma.es/Research/VFwkProg.xml
    </spl:actual_Parameter>
  </spl:instantation>
</spl:PAS>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<spl:SRR xmlns:spl="http://www.lcc.uma.es/ICICS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.lcc.uma.es/ICICS SRR.xsd"
resource="http://www.lcc.uma.es/Research/VFwkProgramme/WP3.pdf">
  <spl:property>
    <spl:property_Name>Object_Type</spl:property_Name>
    <spl:property_Value>report</spl:property_Value>
  </spl:property>
  <spl:property>
    <spl:property_Name>Project_ID</spl:property_Name>
    <spl:property_Value>IST_2001-32446</spl:property_Value>
  </spl:property>
</spl:SRR>
```

Fig. 1. Example Policy, its corresponding PAS and the SRR for a report

Fig. 3 shows the system operation. When the client requests some object from a server it receives the PCO containing it, which runs in the client computer. Before the protected sections of its code can be executed the PCO has to retrieve the corresponding MP (which includes the license that allows the decryption and execution of those protected sections). To do this it sends a request containing the certificate of the public key of the smart card. In case the server from where it was retrieved is the originator of the PCO, it produces the MP for that PCO. Otherwise the server just forwards this request to the originator. Once received, the MP is installed and used to run the protected sections of the PCO.

Policy Specification and Validation. The creation and maintenance of access control policies is a difficult and error prone activity. The Policy Assistant component (which includes the Policy Editor and Semantic Policy Validator) is designed to help security administrators to specify those policies and validate them to find errors. For this purpose, the Policy Assistant provides the administrators with information about the attribute certificates that can be included in the policies, their sources and relation. This information is gathered from SOADs.

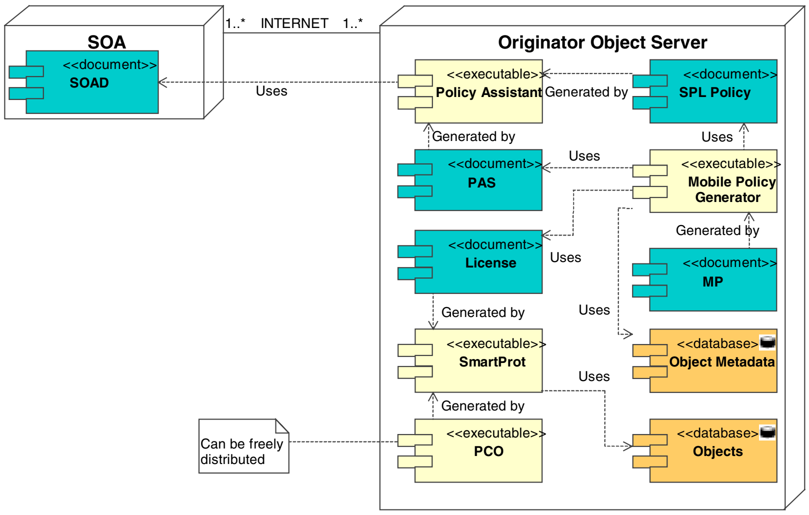


Fig. 2. Overview of the XSCD Infrastructure

The Policy Assistant includes components for the automated validation of policies at different levels. SPL policies are validated syntactically using XML-Schema. Semantic validation is made possible by the use of a specific Semantic Policy Validator that uses the DOM API to parse the document validating it. Finally, policies can be validated taking into account the context where they will be applied.

For instance, consider the case of a research network among several institutions that participate in a common research project. Each participant establishes the access control parameters over the contents they share. Additionally, membership is certified by each institution. Fig. 1 showed an example policy granting access to members assigned to this project of any participant institution, to the commissioner of the project and to the external reviewers assigned to it by the European Union. The institution parameter is instantiated from contextual metadata about the project. On the other hand, the project parameter is instantiated from the SRR corresponding to the object to be accessed. A parameter can represent a complex XML element, as is the case of the institution parameter.

The *Mobile Policy Generator* analyses the semantic metadata available for the target resource contained in the SRR along with other contextual metadata, finds the appropriate PAS and retrieves the necessary SOADs. Using this information, the *Mobile Policy Generator* is able to find the applicable policies. All applicable policies are then analysed and instantiated. Finally, all policies are combined and translated to produce the MP. For the semantic and contextual validation of policies the *Policy Assistant* performs the same process. In this case, after the combination of policies the

Policy Assistant analyses the resulting policy and run some test cases in order to check its consistency. SOADs play a key role in this semantic validation because they provide the information needed to infer all the possible situations where access is granted, enabling the detection of semantically incomplete or incorrect policies.

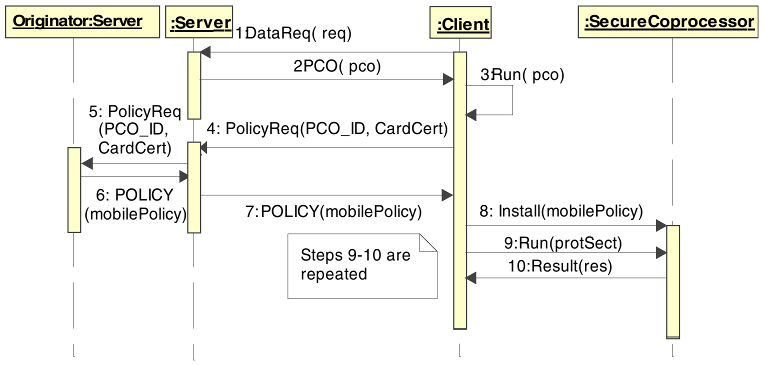


Fig. 3. A scenario for information access

Protected Content Object Generation and Access. The SmartProt system is used to protect software applications. In our prototype system, the protected application (PCO) is a Java applet responsible for the transport of the contents. In other environments PCOs will be implemented accordingly (e.g., CORBA objects, a proxy for web services, etc.). Consequently, the PCO includes the (encrypted) contents to be accessed, the access control enforcement mechanism and a cryptographic link to the MP. PCOs can be distributed and copied freely.

The PCO generation process is independent of the customer secure coprocessor (smart card) and will be performed just once for each piece of mobile software. The first step of the PCO generation consists in the production of a Java applet containing the original object. Then, SmartProt is used to protect this applet. The key generated by SmartProt will be used afterwards to produce the MP.

Mobile Policy Generation. In order to allow that originators of the contents are able to dynamically change the applicable access control policy regardless of the storage location, policy and PCO must be separated. In this way, policies are retrieved from the originating server during the execution of the PCO. This allows a high degree of flexibility giving the originator more control over the application of the policies. For efficiency and flexibility, validity constraints in MPs can be used to control the need for an online access to the originator server. Originators can define certain validity constraints for each policy (based on number of accesses, time, etc. depending on the smart card features). Hence, policies can be cached by clients and used directly while they are still valid. The generation of MPs is a reasonably fast process while the generation of PCOs is slower. Furthermore, PCOs are much more stable than policies. Finally, opposed to PCOs, each MP is specific for a smart card.

Following a user request, the new MP is produced linking the combined SPL policy and the PCO. The MP is obtained and loaded in the card as part of the PCO. When the MP is received by the client smart card, it is decrypted, verified and stored inside the card until it expires or the user explicitly decides to extract it. Once the MP is correctly installed in the card the protected sections of the PCO can be executed, which requires the cooperation of the card containing the MP. The Mobile Policy Generator retrieves and combines all applicable policies to produce the MP. The combination of different policies is usually a difficult task that can result in inconsistent or contradictory policies. To deal with this problem the most common solution is to establish a series of general rules to solve the ambiguous cases. But, in practice, each situation is different. Therefore general rules do not produce good results. Our infrastructure enables the administrator to define rules governing the combination of policies. Additionally, the modular approach and the tools for policy creation, composition and validation facilitate the detection and correction of wrong policies. Because MPs must be processed by smart cards, they are specified using SPL and are later translated into a compact format to be included in MPs. The binding between PCO and the corresponding MP is established by cryptographic means.

5 Conclusions and Future Work

We have presented the XSCD infrastructure for the secure distribution of objects in distributed systems. XSCD extends mobile policies by allowing their execution in untrusted systems and the dynamic and transparent modification of policies. XSCD is based on the SmartProt software protection scheme and the SPL access control scheme. We have introduced mechanisms to seamlessly integrate the external PMI in our infrastructure. The extensive use of XML metadata technologies facilitates the security administration in such environments, and enables important functionalities of the system such as the contextual validation of policies. Furthermore, the combination of policies and the association of policies to objects in a flexible and automated way have been considered. To summarize, XSCD represents a flexible solution for different distributed scenarios and any kind of content, solves the originator-retained-control problem, can be applied regardless of the attribute certification scheme, implements distributed access control management and enforcement mechanisms and allows dynamic modification of policies transparently and efficiently. To the best of our knowledge no other works have been done allowing the semantic validation of policies in distributed environments with separate authorization infrastructures.

A prototype of this system has been implemented for a Digital Library scenario. In such environment, PCOs are implemented using Java applets. *e-gate Cyberflex*™ USB Java smart cards are used as secure coprocessors. The high capacity and the transfer speed of these cards makes possible that the performance of the PCO is very good. A set of techniques, such as temporary authorizations, is used to improve the performance. We are currently working on the application to the CORBA environment. More precisely, we have implemented a Resource Access Decision (RAD) facility based on the XSCD approach.

References

1. Fayad, A., Jajodia, S. *Going Beyond MAC and DAC Using Mobile Policies*. In Proceedings of IFIP SEC'01. Kluwer Academic Publishers. 2001.
2. Sandhu, R.S., E.J. Coyne, H.L. Feinstein and Youman, C.E. *Role-Based Access Control Models*. IEEE Computer, 1996. 29(2): pp. 38-47.

3. Thompson, M., et al., *Certificate-based Access Control for Widely Distributed Resources*. Proceedings of the Eighth USENIX Security Symposium. pp. 215-227. 1999.
4. Chadwick, D. W. *An X.509 Role-based Privilege Management Infrastructure*. Business Briefing. Global Infosecurity 2002. http://www.permis.org/
5. Cryptolope Technology Homepage. http://www-3.ibm.com/software/security/cryptolope/
6. Garcia-Molina, H.; Ketchpel, S.; Shivakumar, N. *Safeguarding and Charging for Information on the Internet*. Proceedings of the Intl. Conf. on Data Engineering. 1998.
7. Intertrust Technologies. http://www.intertrust.com/
8. Yagüe, M. *On the suitability of existing access control and DRM languages for mobile policies.* University of Málaga. Department of Computer Science TR. LCC-2002-7. 2002.
9. Bertino, E., Castano, S., Ferrari, E. On Specifying Security Policies for Web Documents with an XML-based Language. In Proceedings of ACM SACMAT'01. 2001.
10. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P. *A Fine-Grained Access Control System for XML Documents.* In ACM Transactions on Information and System Security (TISSEC), vol. 5, n. 2, May 2002, pp. 169-202.
11. W3C. *XML-Schema*. http://www.w3.org/XML/Schema
12. Bertino, E., Castano, S., Ferrari, E. *Securing XML documents with Author-X*. IEEE Internet Computing, 5(3):21-31, May/June 2001.
13. ITU-T Recommendation *X.509: Information Technology – Open systems interconnection – The Directory: Authentication Framework*, June 1997.
14. ITU-T Recommendation X.509: *Information Technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks*, March 2000.
15. Maña, A., Pimentel, E. *An Efficient Software Protection Scheme*. In Proceedings of IFIP SEC'01. Kluwer Academic Publishers. 2001.
16. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft 30 April 2002. http://www.w3.org/TR/rdf-schema/
17. W3C. *XML-Signature Syntax and Processing*. http://www.w3.org/TR/xmldsig-core/. 2002.

Distributed Key Generation as a Component of an Integrated Protocol[⋆]

Cheng-Kang Chu and Wen-Guey Tzeng

Department of Computer and Information Science,
National Chiao Tung University, Hsinchu, Taiwan 30050
{ckchu,tzeng}@cis.nctu.edu.tw

Abstract. In this paper we discuss the security issue of distributed key generation in a distributed threshold protocol. We identify two subtle flaws in the previously proposed discrete-log based distributed key generation protocols. We propose a discrete-log based distributed key generation protocol that meets the necessary security requirements and has no such flaws.

1 Introduction

In this paper we discuss the security issue of distributed key generation in a distributed threshold protocol. A discrete-log based distributed threshold protocol of computing function f, with threshold t, has two sub-protocols:

1. Distributed key generation (Dkg): a set of players $P_1, P_2, \ldots, P_n$ together generate a public key $y = g^x \bmod p$ such that each honest player P_i holds a share x_i of the secret key x and any $t+1$ or more honest players can recover the secret key x.
2. Threshold function execution (Tfe): a set of t' honest players $P_{i_1}, P_{i_2}, \ldots, P_{i_{t'}}$, $t' \geq t+1$, who were involved in the DKG protocol can compute the function $\sigma = f(y, m, x)$ by combining their partial results $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_{t'}}$, where m is the common input and $\sigma_{i_j} = f'(y, m, s_{i_j})$, $1 \leq j \leq t'$.

The Dkg protocol is executed once only in the initial setting. Then, the qualified players of the threshold protocol execute the Tfe protocol whenever they need to compute such a function distributively.

In previous papers [GJKR96,GJKR99], these two sub-protocols are discussed separately so that their interaction effect is not addressed. The interaction effect between two protocols in a system has been known for years. For example, the auxiliary-input zero-knowledge interactive proof system assumes that the verifier obtains additional information from previous execution of the protocol or other sources. Also, the effect causes security problems even though the executed protocols seem irrelevant [TH99].

[⋆] Research supported in part by the Ministry of Education grant for Excellence 90-E-FA04-1-4, Taiwan, ROC.

We demonstrate flaws about the setting due to such loss of link. We show that even if the DKG protocol meets security requirements, the resultant configuration does not constitute a secure TFE protocol. We show that a corrupted player can get a share of the secret key and an honest player, who follows all steps of the DKG protocol during execution, can actually compute the secret key x.

We believe that there is no reason to let the corrupted (or malicious) player who has shown signs of malice during execution of the DKG protocol to obtain a share of the secret key. Otherwise, when we obtain a $\sigma = f(y, m, x)$, we cannot be sure that it is indeed computed from a set of authorized players. This is because a corrupted (or malicious) player who is disqualified can claim that he is not responsible for computing this σ, but he indeed is involved in. Although it is possible to authenticate the involved players when executing the TFE protocol, it is not practical in most cases and violates the anonymity property sometimes. We identify the following two ingredients to amend the security problem we just mentioned.

1. After executing the DKG protocol, the set of the n' honest players in $QUAL$, who get all broadcast messages, should constitute a valid and secure (n', t)-TFE **even if honest players do additional computation from received messages**.
2. Any player who is not in $QUAL$ **does not get any information about the shares of the secret key** shared by the players in $QUAL$.

Nevertheless, previous DKG protocols only require that the corrupted players **do not get information about the secret key**. As a component of an integrated system, these protocols leak information about the valid shares of the secret key.

Therefore, a discrete-log based distributed threshold protocol with threshold t and to compute function f should consist of the following two sub-protocols with specified properties:

1. Distributed key generation (DKG): a set P of n players together generate a public key $y = g^x \bmod p$. It results in a qualified set $QUAL \subseteq P$ such that each player $P_i \in QUAL$ holds a share x_i of the secret key x and each player $P_j \notin QUAL$ holds no shares of the secret key x.
2. Threshold function execution (TFE): t' players $P_{i_1}, \ldots, P_{i_{t'}}$ of $QUAL$ in which at least $t + 1$ of them are honest can compute the correct result $\sigma = f(y, m, x) = g(y, m, \sigma_{i_1}, \ldots, \sigma_{i_{t'}})$, where $t' \geq t+1$ and $\sigma_{i_j} = f'(y, m, s_{i_j})$ is the partial result computed by P_{i_j}, $1 \leq j \leq t'$. Furthermore, any set of players that contains t or less players from $QUAL$ cannot compute any information about $\sigma = f(y, m, x)$.

In our setting of distributed threshold protocols, we require that the later TFE should meet all security requirements even if some players in $QUAL$ deliberately broadcast malicious information.

Results. In this paper, we propose a discrete-log based distributed key generation protocol. Our DKG protocol meets the necessary security requirements stated above. In particular, the honest players constitutes a secure threshold protocol and the corrupted players get no shares of the secret.

2 Flaws in Previous DKG Protocols

We present a typical discrete-log based DKG protocol [GJKR99] and show how flaws occur in this protocol. Other discrete-log based DKG protocols have the same flaws.

Let the constructed secret key x be in Z_q and the public key be $y = g^x \bmod p$, where $p = 2q + 1$, both p and q are prime, and g is a generator in G_q, which is the order-q subgroup in Z_p^*. In a typical DKG, each player P_i, $1 \le i \le n$, selects a secret value a_i, uses the (t, n)-threshold secret sharing scheme to split a_i into s_{ij}, $1 \le j \le n$, and sends s_{ij} to P_j via a private channel shared between P_i and P_j. Each P_i who gets s_{ji} from P_j, $1 \le j \le n$, computes his own share $x_i = \sum_{j=1}^{n} s_{ji} \bmod q$. Thus, the secret key is $x = \sum_{i=1}^{n} a_i \bmod q$, and the public key is $y = \prod_{i=1}^{n} g^{a_i} \bmod p$, where each P_i publishes $g^{a_i} \bmod p$. If we want the shares to be verifiable, each P_i uses a verifiable secret sharing scheme to share a_i instead. The protocol is stated as follows:

1. Each P_i chooses a random degree-t polynomial
$$f_i(x) = \sum_{j=0}^{t} a_{ij} x^j \bmod q$$
and broadcasts the verification values $A_{ik} = g^{a_{ik}} \bmod p$, $0 \le k \le t$. Let $a_i = a_{i0}$ and $s_{ij} = f_i(j)$, $1 \le j \le n$. P_i sends s_{ij} to P_j via a private channel.
2. Each P_j verifies the received shares s_{ij}, $1 \le i \le n$, by checking
$$g^{s_{ij}} \equiv \prod_{k=0}^{t} (A_{ik})^{j^k} \pmod{p}. \tag{1}$$
If the check fails for index i, P_j broadcasts a complaint against P_i.
3. If P_i receives a complaint from P_j, it broadcasts s_{ij}, which should satisfy Equation (1).
4. Each P_j disqualifies P_i if any of the following conditions holds:
   - It receives complaints against P_i from $t + 1$ or more players.
   - If P_k, $1 \le k \le n$, complains against P_i and s_{ik} revealed by P_i in Step 3 does not match Equation (1).

   Let $QUAL_j$ be the qualification set defined by P_j. Note that all $QUAL_j$ for honest players P_j are equal.
5. Each P_i sets his share as $x_i = \sum_{j \in QUAL_i} s_{ji} \bmod q$ and broadcasts $y_i = g^{a_i} \bmod p$. P_i computes $y = \prod_{j \in QUAL_i} y_j \bmod p$ as the public key.

There are two problems in this protocol. Firstly, if the corrupted P_i does not send a legal s_{ij} to P_j in Step 1, it shall receive a complaint from P_j in Step

2. Even if P_i cannot send a valid s_{ij} that satisfies Equation (1) in Step 3 and shall be disqualified by other players in Step 4, it has already possessed a valid share $x_i = \sum_{j \in QUAL_i} s_{ji}$, which is $F(i) = \sum_{j \in QUAL_i} f_j(i)$. In this case, P_i is disqualified by all players in $QUAL$, but it has a valid share x_i.

Secondly, P_j may file a *false complaint* against P_i in Step 2, that is, P_j actually receives a valid share $f_i(j)$ from P_i, but it still complains against P_i. Then, P_i has to reveal $f_i(j)$ for others to verify in Step 3. We can use the revealed partial information to compute a valid share of the final shared polynomial $F(z)$.

The flaw is as follows. Without loss of generality, we assume that $P_1, P_2, \ldots, P_t$ are the corrupted players and $P_{t+1}, P_{t+2}, \ldots, P_n$ are honest players, where $n \geq 2t+1$. In Step 1, each corrupted P_i, $1 \leq i \leq t$, sends invalid s_{ij} to honest players P_j, $t+1 \leq j \leq n$, such that all honest players P_j would complain against P_i in Step 2. Furthermore, in Step 2, each corrupted P_i, $1 \leq i \leq t$, complains (falsely) against all honest players P_j, $t+1 \leq j \leq n$. Then, in Step 3, each honest player P_j, $t+1 \leq j \leq n$, sends $f_j(i)$, $1 \leq i \leq t$, that satisfies Equation (1).

We can see that all corrupted players are disqualified in Step 4 since all honest players complain against them and there are more than t honest players. The $QUAL$ set is $\{P_{t+1}, P_{t+2}, \ldots, P_n\}$ and the hidden polynomial of the secret sharing is $F(z) = f_{t+1}(z) + f_{t+2}(z) + \cdots + f_n(z)$. Since all honest players are complained, each player P_j, $t+1 \leq j \leq n$, broadcasts $s_{ji} = f_j(i)$, $1 \leq i \leq t$, for verification. Therefore, the shares $F(i) = f_{t+1}(i) + f_{t+2}(i) + \cdots + f_n(i)$, $1 \leq i \leq t$, are publicly known. Furthermore, since each honest player P_j, $t+1 \leq j \leq n$, has an additional share $F(j)$, it can compute $F(z)$. This violates the threshold requirement of the threshold protocol during the execution of TFE.

3 Preliminaries

In this section, we present the models of communication, the honest player, the adversary and the security requirements.

Communication model. We assume that all players are probabilistic polynomial-time Turing machines. They are connected by a round-based and fully synchronous broadcast channel. That is, all players broadcast messages simultaneously in each round and can see all broadcast messages from others without delay and alternation. We also assume that any two players are connected by a private channel such that all other players cannot tape. There is literature discussing how to implement private channels.

Honest player. A player is *honest* if all messages it sends during DKG follow the specification of the protocol. Nevertheless, it may compute information from received messages.

It may be that P_i is corrupted by the adversary, but it shows no signs of malice during execution of the DKG protocol. We treat P_i as honest since it is in the final qualified set and responsible for the TFE protocol.

Adversary. We assume that the adversary is static, which means that the adversary chooses the corrupted (malicious) players before the DKG protocol starts. The computational power of the adversary is modeled as a probabilistic polynomial-time Turing machine. A corrupted player may deviate from the protocol in any way. It can broadcast any message or holds back when required.

Security requirements. A DKG protocol with threshold t allows n participating player to jointly generate a random secret key x, and each player holds a share x_i of the generated secret key x. The protocol also outputs the public key $y = g^x \bmod p$. One can reconstruct the secret key x if he has $t+1$ or more shares. A secure DKG protocol should meet the following *correctness* and *secrecy* requirements.

Correctness. Gennaro, et al. [GJKR99] define the correctness requirements (C1) to (C4). We add the requirement (C5). The combination of (C5) and the secrecy requirement guarantees that no information about the valid shares of the secret key leaks.

(C1) We can construct the unique secret key x on the input of $t+1$ correct shares efficiently.
(C2) Robustness: we can still construct the secret key x from n shares even if up to t shares of them are sent by corrupted players.
(C3) The secret key x is uniformly distributed in Z_q.
(C4) All players get identical public key $y = g^x \bmod p$.
(C5) No disqualified players can hold a correct share of the secret key x.

Secrecy. The adversary learns no information of the secret key x except the public key $y = g^x \bmod p$.

A secure DKG protocol with threshold t is defined as follows.

Definition 1. *A distributed key generation protocol is t-secure if it allows at most t corrupted players and meets the above correctness and secrecy requirements.*

4 An Inefficient Solution

Since disqualified players can obtain valid shares, the qualified players need update their shares so that the shares of the disqualified players become obsolete. An intuitive approach is to use the techniques of proactive secret sharing schemes [HJKY95,HJJ$^+$97] to update shares. However, a corrupted player may act maliciously at any stage. If a malicious player acts during the round of share update, the protocol has to start a new round of share update. Since there are up to t corrupted players, the protocol need t rounds of share update at most. Furthermore, the communication cost of each round of share update is roughly equal to a basic DKG protocol. Therefore, this approach needs cost equivalent to t runs of the basic DKG protocols. It is very inefficient though.

5 Our DKG Protocol

Our DKG protocol is based on the one in [GJKR99]. There are two main different points. First, we add an additional phase for updating the shares of the players in $QUAL$. The share update phase uses the information prepared in the first phase so that no additional rounds are needed. We define two stages of qualified sets: $SQUAL$ (for semi-$QUAL$) and $QUAL$. All players may obtain initial valid shares of the secret key, but only the players in $QUAL$ obtain final valid shares of the secret key. Second, since it is also possible to bias the secret key during the share update phase, we use an additional polynomial to prevent this problem. The three phases of our protocol are shown in Figures 1, 2 and 3.

1. **Secret-key generation:** (in Figure 1)
   In this phase, each player P_i performs the Pedersen-VSS protocol to contribute his share to the secret key. The secret key x is defined uniquely by all honest players. Due to Pedersen-VSS, the secret key x is committed securely in the information-theoretical sense.
   Furthermore, each player P_i generates additional $2(n+1)$ degree-t backup polynomials over Z_q with zero free terms. The first $2n$ backup polynomials $R_i^{(m)}$ and $R_i'^{(m)}$, $1 \le i \le n$, are used to make the shares of dishonest players obsolete in the share update phase. The other two polynomials $R_i^{(0)}$ and $R_i'^{(0)}$ are used to prevent bias of the secret key during the share update phase.
   Each player P_i sends the shares of these polynomials to other players except that P_j, $j \neq i$, does not get the shares $r_{ij}^{(j)} = R_i^{(j)}(j)$ and $r_{ij}'^{(j)} = R_i'^{(j)}(j)$.
   In this phase, a player is disqualified if it is complained by more than t other players or cannot rebut the complaint of other players. The set of players who are not disqualified after this phase is called $SQUAL$ (semi-$QUAL$).
2. **Public-key extraction:** (in Figure 2)
   The players who are not disqualified in the secret-key generation phase publish their partial $y_i = g^{x_i} \bmod p$ of the public key y. Since a player can send out an invalid share of the public key, we disqualify those errant players in this phase. Those players who are in $SQUAL$ and send a valid share of the public key form the set $QUAL$.
   Now, those errant players have shares of the secret key. Those in $QUAL$ need updates their shares so that the shares of other players become obsolete.
3. **Share update:** (in Figure 3)
   The players in $QUAL$ update their shares by adding the backup shares sent in the first phase. That is, each player $P_i \in QUAL$ computes its share as

$$x_i = \sum_{j \in SQUAL} (s_{ji} + \sum_{m \in \{0,\ldots,n\} \setminus QUAL} r_{ji}^{(m)}) \bmod q.$$

   Those dishonest players don't have enough information to update their shares. Therefore, their shares become obsolete.
   Furthermore, each player P_i in $QUAL$ contributes $r_{ij}^{(0)}$, $j \in QUAL$, so that the distribution bias of the secret key won't occur. This completes the protocol.

Generating x:

1. Each player P_i chooses two random polynomials $f_i(z), f'_i(z)$ over Z_q of degree t:
$$f_i(z) = a_{i0} + a_{i1}z + \ldots + a_{it}z^t \quad f'_i(z) = a'_{i0} + a'_{i1}z + \ldots + a'_{it}z^t$$
P_i broadcasts $C_{ik} = g^{a_{ik}}h^{a'_{ik}} \bmod p$ for $k = 0, \ldots, t$, and hands $s_{ij} = f_i(j), s'_{ij} = f'_i(j)$ to player P_j secretly.
2. Again, P_i chooses $2(n+1)$ zero-free-term random polynomials over Z_q of degree t:
$$R_i^{(m)}(z) = b_{i1}^{(m)}z + \ldots + b_{it}^{(m)}z^t \quad R_i'^{(m)}(z) = b_{i1}'^{(m)}z + \ldots + b_{it}'^{(m)}z^t$$
for $m = 0, \ldots, n$. Each P_i broadcasts $D_{ik}^{(m)} = g^{b_{ik}^{(m)}}h^{b_{ik}'^{(m)}} \bmod p$ for $m = 0, \ldots, n$, $k = 0, \ldots, t$, and secretly hands $r_{ij}^{(m)} = R_i^{(m)}(j)$, $r_{ij}'^{(m)} = R_i'^{(m)}(j)$ to player P_j, for $m = 0, \ldots, j-1, j+1, \ldots, n$.
3. Each player P_j verifies the shares he received from others by checking
$$g^{s_{ij}}h^{s'_{ij}} \equiv \prod_{k=0}^{t}(C_{ik})^{j^k} \bmod p \tag{2}$$
and
$$g^{r_{ij}^{(m)}}h^{r_{ij}'^{(m)}} \equiv \prod_{k=0}^{t}(D_{ik}^{(m)})^{j^k} \bmod p \tag{3}$$
for $i = 1, \ldots, n, m = 0, \ldots, j-1, j+1, \ldots, n$. If the check fails for an index i, P_j broadcasts a *complaint* message against P_i.
4. Each player P_i who received a complaint from player P_j broadcasts the corresponding shares (s_{ij}, s'_{ij} or $r_{ij}^{(m)}, r_{ij}'^{(m)}$, for $m = 0, \ldots, j-1, j+1, \ldots, n$).
5. Each player marks as *disqualified* any player that
   – received more than t complaints in Step 3, or
   – answered to a complaint in Step 4 with values that falsify Eq. 2 or Eq. 3.
6. Each player then builds a common set of non-disqualified players $SQUAL$ (Semi-QUAL). (The secret key x is now defined as $x = \sum_{i \in SQUAL} a_{i0} \bmod q$, even though it is not explicitly appeared.)

Fig. 1. Secret-key generation phase

6 Security Proofs

We now prove that our DKG protocol is t-secure, as in Definition 1.

Theorem 1. *The DKG protocol consisting of Figures 1, 2, and 3 is t-secure.*

Proof of Correctness. Assume that $\mathcal{D} = \{0, \ldots, n\} \backslash QUAL$. We prove each correctness requirement individually:

(C1) In the first phase, each player P_i performs $n + 2$ times of Pedersen-VSS and gets the shares $s_{ji}, r_{ji}^{(0)}, \ldots, r_{ji}^{(i-1)}, r_{ji}^{(i+1)}, \ldots, r_{ji}^{(n)}$ from player P_j. We

Extracting $y = g^x \bmod p$:

7. Each player P_i, $i \in SQUAL$, broadcasts $A_{ik} = g^{a_{ik}} \bmod p$ and $B_{ik}^{(m)} = g^{b_{ik}^{(m)}} \bmod p$ for $k = 0, \ldots, t$, $m = 0, \ldots, n$.
8. Each player P_j verifies the values he received from other players in $SQUAL$ by checking
$$g^{s_{ij}} \equiv \prod_{k=0}^{t} (A_{ik})^{j^k} \bmod p \tag{4}$$
and
$$g^{r_{ij}^{(m)}} \equiv \prod_{k=0}^{t} (B_{ik}^{(m)})^{j^k} \bmod p \tag{5}$$
for $i \in SQUAL, m = 0, \ldots, j-1, j+1, \ldots, n$. If the check fails for an index i, P_j *complains* against P_i by broadcasting the shares $s_{ij}, s'_{ij} (r_{ij}^{(m)}, r_{ij}^{\prime(m)})$ that satisfy Eq. 2 (Eq. 3) but do not satisfy Eq. 4 (Eq. 5).
9. For each player P_i who receives at least one valid complaint, the other players reconstruct $f_i(z)$ by running the reconstruction phase of the Pedersen-VSS protocol. Hence, the value A_{i0} can be computed by all players.
10. For all players in $SQUAL$, compute $y = \prod_{i \in SQUAL} A_{i0} \bmod p$.

Fig. 2. Public-key extraction phase

can use these shares to compute the the corresponding polynomial. At the end of the protocol, each honest player P_i obtains
$$x_i = \sum_{j \in SQUAL} (s_{ji} + \sum_{m \in \mathcal{D}} r_{ji}^{(m)}) \bmod q$$
as the share of the secret key x. Thus, for any set $\mathcal{S}$ of $t+1$ correct shares, we can get
$$\begin{aligned} x &= \sum_{j \in SQUAL} a_{j0} + 0 \bmod q \\ &= \sum_{j \in SQUAL} (\sum_{i \in \mathcal{S}} \lambda_i s_{ji} + \sum_{i \in \mathcal{S}, m \in \mathcal{D}} \lambda_i r_{ji}^{(m)}) \bmod q \\ &= \sum_{i \in \mathcal{S}} \lambda_i (\sum_{j \in SQUAL} (s_{ji} + \sum_{m \in \mathcal{D}} r_{ji}^{(m)})) \bmod q \\ &= \sum_{i \in \mathcal{S}} \lambda_i x_i \bmod q, \end{aligned}$$
where λ_i is the Lagrange coefficient for the share of index i. Therefore, the secret key x is uniquely defined and can be efficiently reconstructed from any set of $t+1$ correct shares.

(C2) To show robustness, we need differentiate between correct shares and incorrect ones. We should check the validity of the shares before reconstruction. For each share x_i, we do the following check:

Updating share

11. Each player builds a new set $QUAL$ of players who are in the set $SQUAL$ and receive no valid complaint in Step 8.
12. Each player P_i in $QUAL$ sets his share of the secret as $x_i = \sum_{j \in SQUAL}(s_{ji} + \sum_{m \in \{0,\dots,n\} \setminus QUAL} r_{ji}^{(m)}) \bmod q$

Fig. 3. Share update phase

$$\begin{aligned} g^{x_i} &\equiv g^{\sum_{j \in SQUAL}(s_{ji} + \sum_{m \in \mathcal{D}} r_{ji}^{(m)})} \\ &\equiv \prod_{j \in SQUAL} (g^{s_{ji}} \cdot \prod_{m \in \mathcal{D}} g^{r_{ji}^{(m)}}) \\ &\equiv \prod_{j \in SQUAL} \prod_{k=0}^{t} ((A_{jk})^{i^k} \cdot \prod_{m \in \mathcal{D}} (B_{jk}^{(m)})^{i^K}) \pmod p, \end{aligned}$$

where A_{jk} and $B_{jk}^{(m)}$ are broadcasted in the public-key extraction phase. We drop the shares that fail the check so that incorrect shares cannot influence the reconstruction.

(C3) Since $SQUAL$ is defined in Step 6 of the first phase, the secret key $x = \sum_{i \in SQUAL} a_{i0} \bmod q$ is fixed at the end of the first phase. If one of these constant terms a_{i0}'s uniformly distributes over Z_q, the secret key x also uniformly distributes over Z_q. For an honest player P_i, a_{i0} is uniformly chosen from Z_q and shared via Pedersen-VSS. The secrecy of a_{i0} is unconditional and hence no information about a_{i0} is obtained by the adversary. Therefore, the value a_{i0} uniformly distributes over Z_q without any influence from corrupted players, so does the secret key x and the public key $y = g^x \bmod p$.

(C4) We show that the public key y is indeed equal to $g^x \bmod p$. In Step 10, all players in $SQUAL$ compute

$$\begin{aligned} y &= \prod_{i \in SQUAL} A_{i0} \bmod p \\ &= \prod_{i \in SQUAL} g^{a_{i0}} \bmod p \\ &= g^{\sum_{i \in SQUAL} a_{i0}} \bmod p \\ &= g^x \bmod p. \end{aligned}$$

Since A_{i0} broadcast by player P_i is known to all players, if A_{i0} is equal to $g^{a_{i0}}$ for all $i \in SQUAL$, all players in $SQUAL$ get identical public key $y = g^x \bmod p$. Consider the verification of $A_{ik}(k = 0, \dots, t)$ in Step 8. If no valid complaint is issued against player P_i, the values $A_{i0}, \dots, A_{it}$ define a unique polynomial consistent with at least $t + 1$ shares held by honest players. Therefore, $A_{i0} = g^{f_i(0)} \bmod p = g^{a_{i0}} \bmod p$. Otherwise, if P_i receives valid complaints, the other players take the public values C_{ik}, $k = 0, \dots, t$ and their shares s_{ij} to reconstruct $f_i(z)$ in Step 9. This ensures $A_{i0} = g^{a_{i0}} \bmod p$ as well.

(C5) A *disqualified* player P_i gets s_{ji} and $r_{ji}^{(m)}$ from player P_j for $j \in SQUAL$ and $m \in \{0, \dots, n\} \backslash \{i\}$. Even so, to compute the share x_i, P_i needs to obtain *all* shares $r_{ji}^{(i)}$ for $j \in SQUAL$. Now, we show that P_i can't get all $r_{ji}^{(i)}$ even he colludes with other $t-1$ corrupted players. Consider that each honest player P_j in $SQUAL$ performs a Pedersen-VSS on $R_j^{(i)}(z)$ in Step 2 and he doesn't send $R_j^{(i)}(i)$ to player P_i. According to Pedersen-VSS, it requires $t+1$ or more shares to reconstruct the polynomial, and hence the share $R_j^{(i)}(i)$. Consequently, P_i can't get any information about $r_{ji}^{(i)} = R_j^{(i)}(i)$ since there are at most t corrupted players.

Proof of Secrecy (sketch). We construct a simulator *SIM* in Figure 4 and show that the view of the adversary $\mathcal{A}$ in the following two conditions are the same:

- $\mathcal{A}$ interacts with *SIM* on input y.
- $\mathcal{A}$ interacts with the honest players in the real run of the protocol with output y.

Let us discuss the simulator *SIM* first. Without loss of generality, we assume that $\mathcal{B} = \{1, \dots, t'\}$ is the set of players controlled by the adversary and $\mathcal{G} = \{t'+1, \dots, n\}$ is the set of honest players, where $t' \le t$. After Step 1, the set *SQUAL* is well-defined. Since $|\mathcal{G}| > t$, *SIM* can interpolate all polynomials held by malicious players in *SQUAL*. Thus, *SIM* knows all polynomials $f_i(z)$, $f_i'(z)$, $R_i^{(m)}(z)$, and $R_i'^{(m)}(z)$ for $i \in SQUAL$, $m = 0, \dots, n$ In order to simulate the run of the protocol that ends with the output y, we choose some suitable values A_{ij}'s that satisfy

$$y \equiv \prod_{i \in SQUAL} A_{i0} \bmod p.$$

In Step 2, we let

$$A_{n0}^* = y \cdot \prod_{i \in SQUAL \backslash \{n\}} (A_{i0}^{-1}) \bmod p$$

and set A_{nk}^* to the appropriate values for $k = 1, \dots, t$. Since there is no output or communication in the share update phase of the real protocol, we need not care about the share update phase here.

Note that the polynomials $R_i^{(m)}(z), R_i'^{(m)}(z)$ and its corresponding values $r_{ij}^{(m)}$ and $r_{ij}'^{(m)}$ are all generated in Step 1 of *SIM*. In other words, these values are generated in the same way as the real protocol runs. No related output is produced in later phases. Hence we omit the description of these values in the proof below.

Now, we show that *SIM* outputs a probability distribution *identical* to the real run of the protocol. The polynomials f_i^* and $f_i'^*$, $i \in \mathcal{G} \backslash \{n\}$, output by *SIM*,

Simulator *SIM*

Input: public key y

1. Simulate the honest players $P_{t'+1}, \ldots, P_n$ to perform the first phase of the protocol.
2. Perform all steps in the public-key extraction phase on behalf of the honest players, except that player P_n broadcasts A^*_{nk}, instead of A_{nk}:
   - $A^*_{n0} = y \cdot \prod_{i \in SQUAL \setminus \{n\}} (A_{i0}^{-1}) \bmod p$.
   - $A^*_{nk} = (A^*_{n0})^{\lambda_{k0}} \cdot \prod_{i=1}^{t} (g^{s_{ni}})^{\lambda_{ki}}$ for $k = 1, \ldots, t$, where λ_{ki}'s are the Lagrange interpolation coefficients.

Fig. 4. Simulator for our DKG protocol

are set to f_i and f'_i. The polynomial f^*_n is defined by the values $f^*_n(j) = f_n(j)$, $j = 1, \ldots, t$, and $f^*_n(0) = \log_g A^*_{n0}$, where A^*_{n0} is evaluated from the random and uniformly distributed input y. Then, f'^*_n is immediately defined via f^*_n and C_{nk}, $k = 0, \ldots, t$, that is, $f'^*_n(z) = (\log_g h)^{-1} \cdot (f_n(z) - f^*_n(z)) + f'_n(z) \bmod q$. So, it can be seen that for $i \in \mathcal{G}$, f^*_i and f'^*_i are all in the appropriate distribution. Furthermore, the coefficients of these polynomials committed to the values C_{ik} and A_{ik}, $0 \leq k \leq t$, are also in the right distribution.

Thus, the view of the adversary consists of values $f_i(j), f'_i(j)$ (*SIM* sends), and values C_{ik}, A_{ik} (*SIM* broadcasts), $i \in \mathcal{G}, j \in B, k = 0, \ldots, t$, in the two conditions we describe above are the same. This ensures secrecy.

References

GJKR96. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In *Proceedings of Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *LNCS*, pages 354–371. Springer-Verlag, 1996.

GJKR99. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Proceedings of Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *LNCS*, pages 295–310. Springer-Verlag, 1999.

HJJ$^+$97. Amir Herzberg, Markus Jakobsson, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 100–110. ACM, 1997.

HJKY95. Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Proceedings of Advances in Cryptology - CRYPTO '95*, volume 963 of *LNCS*, pages 339–352. Springer-Verlag, 1995.

Ped91. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of Advances in Cryptology - CRYPTO '91*, volume 576 of *LNCS*, pages 129–140. Springer-Verlag, 1991.

TH99. Wen-Guey Tzeng and Chi-Ming Hu. Inter-protocol interleaving attacks on some authentication and key distribution protocls. *Information Processing Letters*, 69:297–302, 1999.

A Secure Agent-Mediated Payment Protocol

Xiaolin Pang, Kian–Lee Tan, Yan Wang, and Jian Ren

Department of Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
Republic of Singapore
{pangxiao,tankl,ywang,renjian}@comp.nus.edu.sg

Abstract. While software agents have been employed in payment protocols, they are largely passive entities, i.e., they participate in the payment protocol but do not make decision. In this paper, we propose an agent-assisted payment protocol called LITESET/A+ that empowers the payment agent (PA) to perform encryption operation for its owner. This is realized by introducing a Trusted Third Party (TTP) in the payment system based on the SET protocol (Secure Electronic Transaction) and a novel signcryption-threshold scheme. In LITESET/A+, the PA and TTP collaborate together to ensure the same level of security as the SET specification. At the same time, with the signcryption-threshold scheme, the PA is more flexible and autonomous during trading.

1 Introduction

Although the Internet is now becoming an important environment for e-commerce, the public is still wary of buying goods and also paying for them on-line. For example, there is concern that credit card information, when submitted on-line, may be eavesdropped despite the fact that very few of those attacks have actually succeeded. Even the deployment of secure servers, based on protocols such as SSL [1] or S-HTTP [2], is not considered to be secure enough to protect the credit card information since it is deposited on the sever, where it can potentially be read by anyone who have access to the server.

To protect the user's credit card information over open network like the Internet, the SET (Secure Electronic Transaction) protocol [3] has been developed mainly by credit card industry such as VISA and MasterCard, in association with major software and cryptography companies. SET provides many important secure properties – such as authentication of participants, confidentiality of information, integrity of data and non-repudiation, etc. In the payment system, each participant can only obtain the information that is necessary for it to perform its own function. For example, the merchant never gets the buyer's credit card information, and the financial institution authorizing the transaction never knows the details of the purchase like the nature of the products, quantities, etc. Nevertheless, SET is a complex protocol and may be unsuitable under some technical conditions.

Several extensions of the SET protocol have been proposed. The SET/A [4] protocol dispatches an agent to the merchant server so that all operations can be performed there. In this way, the user no longer needs to connect to the Internet when the transaction is running. The SET/A+[5] protocol removes the requirement of SET/A for a secure agent execution environment by adding a Trust Verification Center (TVC) in the payment system. TVC keeps the sensitive information and provides verification services for cardholder and merchants. But the payment agent of SET/A+ is not authorized to certain functions, e.g., to ensure that some operations are performed in a non-repudiable way or to encrypt certain important information.

To improve the flexibility of mobile agents and the efficiency of SET/A+ protocol, a LITESET/A+ protocol based on LITESET system (Light-Weight Secure Electronic Transaction) is proposed in this paper. By using a new cryptography technique –signcryption [6], LITESET protocol reduces the heavy computation and message overhead in the employment of SET, the implementation of which is based on traditional RSA signature and encryption scheme [7]. In LITESET/A+ protocol, we introduce a Trusted Third Party (TTP) and a newly proposed signcryption-threshold scheme. TTP and payment agent collaborate together to protect the sensitive information such as credit card information and the signature private key carried by the agent during the whole payment process.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 describes a proposed signcryption-threshold scheme, and Section 4 presents the LITESET/A+ payment protocol. Security issues are discussed in section 5. Finally, Section 6 concludes this work.

2 Related Work and Background

2.1 Related Work of Payment Protocol

2.1.1 SET Protocol

There are five participants in SET protocol: cardholder, issuer, merchant, acquirer and payment gateway. Each participant possesses two distinct asymmetric key pairs. One is used for performing encryption and decryption function and its public key is authenticated by key- exchange certificate (C_K). The other is used for generation and verification of signatures and its public key is authenticated by signature certificate (C_S).

Since the phase on purchase request is the core of the whole transactions, only this phase is emphasized and described in detail in this work (see Fig. 1).

2.1.2 LITESET Protocol

Although SET is treated publicly as one of the important protocols for electronic payments, a straightforward implementation incur significant computation and message overhead, primarily because it uses traditional techniques such as RSA digital signature and encryption scheme [7]. LITESET [6], a lightweight secure electronic transaction protocol, improves the efficiency by using signcryption– a new cryptographic scheme. For the same level of security as the SET specification, LITESET shows a 53.7% reduction in the computational time in message generation/verification and a 79.9% reduction in communication overhead [6].

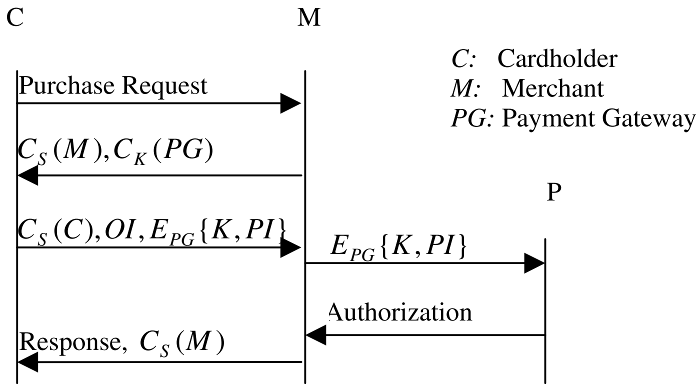


Fig. 1. SET purchase request transaction

2.1.3 SET/A Protocol

Since SET is very complex and may not be suitable under some technical conditions, SET/A protocol [4] is proposed to make SET adaptable to the mobile computing environments. Based on the principles used in purchase phase of SET, SET/A improve its performance only by adding a mobile agent for the cardholder to fulfill payment transaction. Also it is a practical technique since the cardholder need not frequently connect to Internet during the whole transaction phase. SET/A performs the same function of transaction as that in SET except that mobile agent of SET/A replaces the cardholder of SET in the purchase phase. However, SET/A protocol also has its drawbacks: it has to rely on a secure execution environment or hidden computation on the merchant server since all the critical data carried by agent are deposited on it.

2.1.4 SET/A+ Protocol

SET/A+ protocol [5] removes the security requirement of agent's running environment on merchant server by adding a Trust Verification Center (TVC) in the payment system. The TVC keeps the sensitive information and charges cardholders or merchants by providing verification service. However, the agents are limited in their functionalities. For example, an agent cannot sign and perform encryption for the owner during trading (since it requires the secret key of the owner).

3 Signcryption-Threshold Scheme

In this paper, a signcryption-threshold scheme is proposed, which is based on the special case of shamir-threshold scheme *(t=w)* that protects the secret K by distributing w secret shares from the secret to t users. We also extend the scheme to work with signature-only mode of signcryption scheme, namely signature-threshold scheme.

1. In the scheme, there are three public parameters (p, q, g), two public hash functions (hash, KH) and one pair of encryption/decryption algorithms (E, D).

The notations of the above parameters are:

p- a large prime
q- a large prime factor of p-1
g- an integer chosen randomly from [1,…,p-1] with order q modulo p
hash- a one-way hash function whose output has ,say, at least 128 bits
KH - a keyed one-way hash function, which is for secure message authentication.
(E, D)- the encryption and decryption algorithms of a private key cipher

2. Each entity in the payment system owns a key pair. For example, entity A has a key pair (y_a, x_a), where $y_a = g^{x_a} (\bmod p)$ and the private key x_a is chosen uniformly at random from [1,…,q-1] .

3. To protect one entity's private key, we assume several parties share it using the special case of shamir-theshold scheme (t=w). For example, to protect entity A's signature private key x_a, t parties $(A_1, A_2, ..., A_t)$ will share the private key and have secret shares $x_{a_1}, x_{a_2}, ..., x_{a_t}$ respectively, where $x_a = x_{a_1} + x_{a_2} + ... + x_{a_t} (\bmod p)$.

4. We are ready to describe the signcryption-threshold scheme:
In the scheme, there are several participants: sender A, recipient B and t sharing-parties $(A_1, A_2, ..., A_t)$ to share the secret for A.

Sender A:	Private key - x_a
	Public key - $y_a, (y_a = g^{x_a} \bmod p)$
Sharing-Parties:	The private key x_a is shared by t parties ($A_1, A_2, ..., A_t$), each of which has the secret share $x_{a_1}, x_{a_2}, ..., x_{a_t}$ respectively and $x_a = x_{a_1} + x_{a_2} + ... + x_{a_t} (\bmod p)$.
Recipient B:	Private key - x_b
	Public key - $y_b, (y_b = g^{x_b} \bmod p)$

Two modes of the scheme (i.e. signcryption-threshold scheme and signature-threshold scheme) are described in detail below:

(1) Signcryption-threshold on a message m (Sender: $A_1, A_2, ..., A_t$, Recipient: B)

a) Signcrypting m by sharing-parties ($A_1, A_2, ..., A_t$):

$$(k_1, k_2) = hash({y_b}^x \bmod p) \tag{1}$$

$$c = E_{k_1}(m) \tag{2}$$

$$r = KH_{k_2}(m) \tag{3}$$

$$s_i = x/(r + x_{a_i}) \bmod q \tag{4}$$

Where equation (1) uses a hash function- $hash({y_b}^x \bmod p)$ and splits the result into k_1 and k_2 of appropriate length. The key k_1 is used to encrypt message m by equation (2) and a key hash function - $KH_{k_2}(m)$ - performs hash function on message m using key k_2 by equation (3). From the results of (2) and (3), each party computes $s_i = x/(r + x_{a_i}) \bmod q$, where $1 \le i \le t$, and generates a part of signature respectively.

In above equations, x is a random number chosen from [1,…,q-1] , but its value keeps the same for each sharing- party. The difference of signcryption operation among each party is the last equation (4) used to produce signature s_i because the secret share x_{a_i} may be different to each party.

After the signcrypted text (c, r, s_i) is generated, it is sent to B by each party.

b) Unsigncryption of (c, r, s_i) by B involves the followings:

$$(k_1, k_2) = hash((y_a \cdot g^{nr})^{(\sum_{i=1}^{t} s_i^{-1})^{-1} \cdot x_b} \bmod p) \tag{5}$$

$$m = D_{k_1}(c) \tag{6}$$

$$KH_{k_2}(m) \overset{?}{=} r \tag{7}$$

Where equation (5) performs the public hash function-

$hash((y_a \cdot g^{nr})^{(\sum_{i=1}^{t} s_i^{-1})^{-1} \cdot x_b} \bmod p)$ to recover (k_1, k_2) from input signature parts (c, r, s_i) sent by each sharing-party. Then B decrypts c using key k_1 to get message m by equation (6) and verifies it by checking if $KH_{k_2}(m) = r$.

(2) Signature-threshold on a message m (Sender: $A_1, A_2, ..., A_t$, Recipient: B)

a) Signature part (r, s_i) on a message m generated by sharing-parties ($A_1, A_2, ..., A_t$):

$$r = hash(g^x \bmod p, m) \tag{8}$$

$$s_i = x/(r + x_{a_i}) \bmod q \tag{9}$$

Where equation (8) performs hash function- $hash(g^x \bmod p, m)$ and gets r, then $s_i = x/(r + x_{a_i}) \bmod q$ is computed by each party to get the A's signature part, where $1 \le i \le t$.

After the signature part (m, r, s_i) is generated, it is sent to B by each party.

b) Verification of signature (r, s_i) of A can be performed as follows:

$$v = hash(y_a \cdot g^{nr})^{(\sum_{i=1}^{t} s_i^{-1})^{-1}} \bmod p \tag{10}$$

$$hash(v, m) \overset{?}{=} r \tag{11}$$

Where equation (10) performs public hash function-

$v = hash(y_a \cdot g^{nr})^{(\sum_{i=1}^{t} s_i^{-1})^{-1}}$ mod p and B checks if $hash(v, m) = r$ with equation (11).

4 LITESET/A + Protocol

In this section we present the payment protocol LITESET/A+, where the agent acting on behalf of the cardholder is much more capable than that in SET/A+. Once the agent is authorized to buy certain kind of product, all further activities such as appropriate decision of buying, negotiation and signing for certain action, will be performed without the cardholder's assistance.

We propose the LITESET/A+ protocol by adding a trusted third party (TTP) in LITESET based on the proposed signcryption-threshold scheme. The TTP can be assumed as a trusted agent to perform partial functions for the cardholder. The TTP also contributes to support non-repudiation between cardholder and merchant. In our approaches, payment agent and TTP share the randomly generated symmetric key K for the encryption of credit card information (i.e., $E_k(PI)$), which is based on the special case of shamir-threshold scheme (t=w=2). At the same time, they also share the signature private key x_{SC} for signing messages on behalf of cardholder, which is based on the special case of the signcryption-threshold protocol (t=w=2). They share the two kinds of keys. Any one of them cannot retrieve the secret keys and therefore the credit card information and private signature key could be protected.

The notation of the certificate, signature key pair and key-exchange key pair of each participant is illustrated in Table 1.

Table 1. Two Distinct Key Pairs & Certificates of Participants

Name	Signature Certificate, Public key & Private Key	Key-exchange certificate, Public key & Private Key
Cardholder (C)	$C_S(C), y_{SC}, x_{SC}$	$C_K(C), y_{KC}, x_{KC}$
Merchant (M)	$C_S(M), y_{SM}, x_{SM}$	$C_K(M), y_{KM}, x_{KM}$
Payment Gateway (PG)	$C_S(PG), y_{SPG}, x_{SPG}$	$C_K(PG), y_{KPG}, x_{KPG}$
Trust Third Party (TTP)	$C_S(TTP), y_{STTP}, x_{STTP}$	$C_K(TTP), y_{KTTP}, x_{KTTP}$
Payment Agent (PA)	$C_S(PA), y_{SPA}, x_{SPA}$	$C_K(PA), y_{KPA}, x_{KPA}$

4.1 Secret-Sharing of Symmetric Key K

The symmetric key K used to encrypt the credit card information is divided into: KS_{PA} - the shared secret of payment agent, and KS_{TTP} - the shared secret of TTP. The construction can be seen in the following formula:

$$\begin{aligned} KS_{PA} &= K - x + R \\ KS_{TTP} &= x + I_C + T \end{aligned} \tag{12}$$

Where x, R are random numbers chosen from [1,…,q-1] , I_C is the transaction identifier assigned by cardholder and T is timestamp.

Reconstruction of the key K is described in Section 4.3.

4.2 Secret-Sharing of Cardholder's Signature Private Key x_{SC}

The payment agent and TTP share cardholder's signature private key x_{SC} are based on shamir-threshold scheme. See formula (13):

$$\begin{aligned} xs_{PA} &= x^{'} \\ xs_{TTP} &= x_{SC} - x^{'} \end{aligned} \tag{13}$$

Where $x^{'}$ is a random number chosen from [1,…,q-1] .

Verification of the cardholder's signature can be obtained from the above sign-cryption-threshold scheme in Section 3.

4.3 Procedure of Payment Transaction

In this section, we focus on the purchase request in LITESET/A+ protocol and the procedure is sketched in Fig. 2.

Following is the detailed procedure of payment transaction in LITESET/A+:

Step1. The cardholder C builds a purchase description with the same elements of SET. Then an agent A(C) on behalf of the cardholder is dispatched to search the required goods, perform negotiation with merchants and pay for goods finally.

The cardholder's agent contains: the cardholder's signature certificate ($C_s(C)$), the request information, the information for TTP, the digest of payment instructions (PI)(i.e. H (PI)) and the encrypted payment instructions ($E_k(PI)$).

The information for TTP includes:

$$(k_1, k_2) = hash\ (y_{KTTP}^{x} \bmod p)$$

$$c = E_{k_1}(m \,||\, KS_{TTP} \,||\, xs_{TTP})$$

$$r = KH_{k_2}(m \,||\, KS_{TTP} \,||\, xs_{TTP} \,||\, Customer\_Info)$$

$$s = x/(r + x_c) \bmod q$$

Notes:

- m is the message for TTP, including the transaction identifier I_C assigned by cardholder , the merchant host name and the time stamp T.

- Customer_Info contains the data of $C_s(C)$, $hash(Constraint)$, where Constraint is the cardholder's buying constraint on mobile agent, such as certain goods' brand, the goods' quantities, the maximum money allowed to be spent, etc.

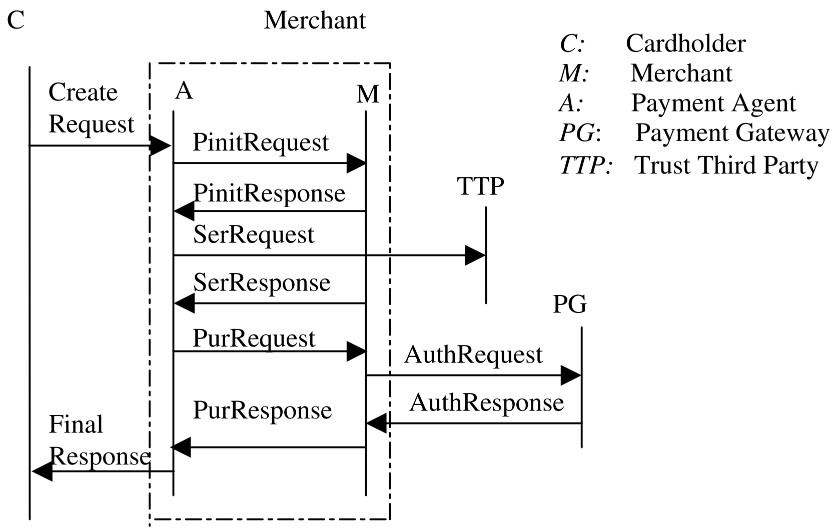


Fig. 2. LITESET/A+ purchase request transaction

Then, the agent on behalf of the cardholder is dispatched to one of merchant servers.

Step2. After the agent finishes searching and negotiation phase, it decides to pay required goods of certain merchant M. To make purchase and payment, it arrives at the merchant server and submits cardholder's signature certificate $(C_s(C))$. Merchant M verifies them. If correct, it supplies the agent an execution environment.

Step3. The agent residing on the merchant server then sends information (c, r, s) to TTP (See Step1) and also the purchase initial request (PInitReq) to merchant locally including the card brand name. It also asks the merchant to respond with a copy of the payment gateway's key-exchange certificate $C_k(PG)$.

Step4. Upon receiving the information from payment agent, TTP unsigncrypts it as: $(k_1, k_2) = hash((y_C \cdot g^r)^{s \cdot x_{KTTP}} \bmod p$ and get m, KS_{TTP}, xs_{TTP} from $D_{k_1}(c)$. TTP accepts only if $KH_{k_2}(m, KS_{TTP}, xs_{TTP}, Customer\_Info) = r$ and it keeps Customer_Info, m, r and s as the non-repudiation evidence.

Step5. When merchant receives the request, it returns a purchase initial response (PinitRes), containing a signed message with its signature certificate $C_s(M)$, the payment gateway key-exchange certificate $C_k(PG)$, and a unique transaction identifier (I_M) to the payment agent.

Step6. The agent verifies the merchant's certificate $C_s(M)$ and the signature. After verification, agent generates the order information (OI) and the half dual signature of the cardholder on them (i.e. $xs_{PA}[H(H(PI) \| H(OI))]$), etc. The payment agent sends message to the TTP including the payment gateway key-exchange certificate $C_k(PG)$, $C_s(M)$ and also the shared secret key KS_{PA} used to protect the credit card information.

Step7. Upon obtaining the above information, TTP generates another half dual signature of the cardholder on the OI and the PI (i.e. $xs_{TTP}[H(H(PI) \| H(OI))]$) and reconstructs the symmetric key K: $KS_{TTP} + KS_{PA} = K + I_C + T + R$. And TTP keeps $C_s(M) + I_M$ as the non-repudiation service. It then gives information to payment agent including y_{KPG} $(K + I_C + T + I_M + R)$.

Step8. When the payment agent obtains the above information, the agent creates the digital envelope E_{PG} for the payment gateway: $E_{PG} = \{y_{KPG}(K + I_C + I_M + T + R), I_C, I_M, T, R, E_K(PI)\}$. And the purchase request (PReq (PIData, OIData)) for the merchant includes: $C_s(C), H(PI), OI, xs_{PA}[H(H(PI) \| H(OI))], xs_{TTP}[H(H(PI) \| H(OI))]E_{PG}$

Step9. After receiving the purchase request, the merchant M checks the cardholder's certificate $C_s(C)$ and verifies the dual signature of the cardholder (i.e. $xs_{PA}[H(H(PI) \| H(OI))], xs_{TTP}[H(H(PI) \| H(OI))]$) with H (PI) and OI. It is obvious that merchant M cannot get PI during the verification. If verification succeeds, merchant M sends the rest information (i.e. $E_{PG}, xs_{PA}[H(H(PI) \| H(OI))], xs_{TTP}[H(H(PI) \| H(OI))], C_S(C)$) to payment gateway for authorization.

Step10. The payment gateway obtains the symmetric key K from E_{PG} if I_C, I_M, T are not abused and hereby gets PI. Then the payment gateway verifies $C_S(C)$ and the dual signature (i.e. $xs_{PA}[H(H(PI) \| H(OI))], xs_{TTP}[H(H(PI) \| H(OI))]$) by applying PI and H (OI). If all these are correct, payment gateway sends authorization response.

Step11.After processing the order, the merchant generates and signs a purchase response, and sends it to the agent along with its signature certificate. If the payment is authorized, the merchant will fulfill the order, by delivering the products bought by the cardholder.

Step12.The agent verifies the merchant signature certificate, checks the digital signature of the response, and then returns back to its owner. The owner takes any appropriate actions based on its contents.

From the above we can see that LITESET/A+ has only two steps more than SET/A+. In LITESET/A+, the agent not only has the function of doing payment for cardholder, it also searches goods information and negotiates with merchants in pervious steps. And since the agent and TTP share the signature private key of the cardholder, they generate half of the dual signature on OI and PI for cardholder respectively. This inevitably adds steps in payment process. We can see it doesn't significantly affect the efficiency of LITESET/A+ but can improve the security level.

5 Security Issues

5.1 Security of Proposed Signcryption Scheme

The security of Signcryption is based on the discrete logarithm problem. It satisfies unforgeability, non-repudiation and confidentiality conditions [12][13]. And the security of Shamir-threshold scheme relies on the provable assumptions [10][11]. As for the proposed signcryption scheme that is based on Shamir-threshold scheme, each sharing party won't know secret shares except its own share. For example, when an attacker wants to forge the cardholder's signature by performing signature-threshold scheme, he/she must get each share part from each sharing-party and it is almost impossible except they conspire together.

5.2 Protection of the Credit Card Information

Since the agent perhaps runs on a hostile server, the sensitive information such as credit card information should not be disclosed on any other environment except to payment gateway in LITESET. In our approach, the payment agent and the TTP will collaborate together to protect the credit card information to fulfill the payment transaction. The proposed LITESET/A+ protocol achieves such objective by using signcryption-threshold scheme. Even TTP and merchant taking part in the process cannot retrieve encrypted credit card information. For example, when TTP reconstructs the secret information of symmetric key K ($KS_{TTP} + KS_{PA} = K + I_C + T + R$), it can only obtain $K + I_C + T + R$ and cannot obtain K since payment agent keeps the random number R. And also when merchant gets the E_{PG}, it cannot decrypt the encrypted PI and y_{KPG} $(K + I_C + T + I_M + R)$.

5.3 Protection of the Private Key Carried by Agent

To protect the owner's private key carried by agent, Kotzanikolaou [8] presented a solution to the hostile host by encrypting a signature function based on RSA signature scheme. Although the scheme protects cardholder's private key successfully, it doesn't ensure non-repudiation property. Since the signature can be computed by any party, the merchant server can repudiate the cardholder's signature generation later. Another solution [9] is to issue proxy certificate [14][15] to an agent, which is based on delegation type as partial delegation (issuing a new key pair to the agent), and the agent will sign for its owner using the new private key. But this solution has obvious limitations since the private key of the agent can also be attacked at any time. In this paper, our approach to solve this problem is that agent and TTP will share the cardholder's private key to protect the signature key from detecting. The proposed signcryption-threshold scheme is adopted to protect owner's private key. Agent has one share of the private key xs_{PA} (i.e. $x^{'}$) and TTP owes the other share xs_{TTP} (i.e. ($x_{SC} - x^{'}$)). Even the TTP cannot retrieve the private key except agent and TTP conspire together. It can only be known to its owner. In this proposed approach, both agent and TTP perform signing or signcryption based on threshold-signcryption scheme for cardholder when needed.

6 Conclusions

In this paper, we have proposed a LITESET/A+ payment protocol that is based on Shamir-threshold scheme and a newly proposed signcryption-threshold scheme, and employs a Trusted Third Party. It allows an agent to perform sign/signcryption operations for its owner during purchase. The mobile agent automatically roams among some on-line merchants, finds the most suitable sites, negotiates with them and then purchases satisfied goods by using LITESET/A+ protocol on behalf of cardholder. In this approach, a cardholder need not frequently connect to Internet and only need to wait for the purchase response at last. LITESET/A+ protocol is also computationally efficient as it is based on signcryption-threshold scheme. In view of above discussions, we believe LITESET/A+ is acceptable for both cardholder side and merchant side.

The future work will consider optimizing the secure payment protocol and achieving the combination of the payment protocol with the searching information and negotiation protocol in reality. The agent will automatically and independently fulfill the whole business transaction for its owner.

References

1. URL: http://www.netscape.com/eng/ssl3/ssl-toc.html
2. URL: http://www.terisa.com/shttp/current.txt
3. VISA INTERNATIONAL, and MASTERCARD INTERNATIONAL. Secure Electronic Transaction (SET) Specification. Version 1.0, (1997)
4. Romao, A., M. M. da. Sliva: " An agent-based secure Internet payment system for mobile computing", TREC'98, LNCS 1402, (1998) 80-93
5. Yi, X., Siew, C. K., Wang, X. F., Okamoto, E.: "A secure Agent-based Framework for the Internet Trading in Mobile Computing Environments", in Distributed and Parallel Databases, 8. (2000) 85-117
6. Hanaoka, G., Zheng, Y., Imai, H.: "LITESET: a light-weight secure electronic transaction protocol", Proc. ACISP'98, Lecture Notes in Computer Science, vol.1438. Springer-Verlag. (1998) 215-226
7. Rivest, R. L., Shamir, A., Adleman, L.: " A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, 21. (1978) 120-126
8. Kotzanikolaou, P., Burmester, M., Chrissikopoulos, V.: "Secure Transactions with Mobile Agents in Hostile Environments", ACISP 2000,LNCS 1841. (2000) 289-297
9. Romao, A., Sliva, M. M. da.: " Secure mobile agent digital signatures with proxy certificates", E-Commerce Agents, LNAI 2033. (2001) 206-220
10. Stinson, D. R.: " Secret Sharing Schemes", Cryptography-Theory and Practice, CRC Press. (1995) 326-331

11. Menezes, A., Van Oorschot, P., Vanstone, S.: "Threshold Schemes", Handbook of Applied Cryptography, CRC Press. (1996)
12. Zheng, Y.: "Signcryption and Its Applications in Efficient Public Key Solutions", Information Security Workshop (ISW '97), Springer-Verlag, LNCS 1397. (1998) 291-312
13. Zheng, Y.: "Digital Signcryption or How to Achieve Cost (Signature& Encryption)<<Cost (Signature)+Cost (Encryption), in Advances in Cryptology-CRYPO'97, vol 1294. Springer-Verlag. (1997) 165-179
14. Mambo, M., Usuda, K., Okamoto, E.: "Proxy Signatures for Delegating Signing operation", Proc.Third ACM Conf. On Computer and Communications Security. (1996) 48-57
15. Kim, S., Park, S., Won, D.: "Proxy signatures, Revisited", Proc. Of ICICS'97, International Conference on Information and Communications Security, LNCS1334, Springer-Verlag. (1997) 223-232

Tensor Transform of Boolean Functions and Related Algebraic and Probabilistic Properties

Alexander Kholosha and Henk C.A. van Tilborg

Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513
5600 MB Eindhoven, The Netherlands
{A.Kholosha,H.C.A.v.Tilborg}@tue.nl

Abstract. We introduce a tensor transform for Boolean functions that covers the algebraic normal and Walsh transforms but which also allows for the definition of new, probabilistic and weight transforms, relating a function to its bias polynomial and to the weights of its subfunctions respectively. Our approach leads to easy proofs for some known results and to new properties of the aforecited transforms. Finally, we present a new probabilistic characteristic of a Boolean function that is defined by its algebraic normal and probabilistic transforms over the reals.

Keywords: cryptography, key-stream generator, Boolean function, tensor transform, Walsh transform, probabilistic properties.

1 Introduction

The two most common building blocks for key-stream generators are the nonlinear filter generator and the nonlinear combination generator [1]. They correspond respectively to a nonlinear transformation applied to several phases of the same linear feedback shift register (LFSR) or to the outputs of several independent LFSR's. The nonlinear transformation can be represented by a Boolean function and the security of the key-stream generators heavily relies on the specific qualities of this function. If the function is not chosen properly then the whole system is susceptible to different types of correlation [2] and linear [3] attacks.

It is currently generally accepted that secure Boolean function to be used in a key-stream generator must satisfy the following properties: balancedness, high nonlinearity, sufficiently high algebraic degree (this should hold for each individual variable), optimized with correlation properties. These conditions are necessary, although it is not clear if they are sufficient to resist all kinds of attacks. The algebraic degree of a Boolean function is the degree of its algebraic normal form (ANF), balancedness, nonlinearity, and correlation properties are defined by its Walsh transform [4]. Thus, the algebraic normal and Walsh transforms of a Boolean function define the most important cryptographic characteristics of the function. The objective of this paper is to generalize known transforms of Boolean functions and develop new ones that would provide efficient means for analyzing security of these functions.

In Sect. 2 we describe the general basis for a tensor transform of Boolean functions. Special cases of this approach provide easy proofs for some known and new relations in the theory of algebraic normal and Walsh transforms. We also propose a new type of tensor transform, the probabilistic transform, giving an important insight in certain probabilistic properties of Boolean functions that are discussed in Sect. 3. Another new type of tensor transform that we propose, is the weight transform. It relates a Boolean function to the weights of its subfunctions. It is proved that coefficients of the ANF of a Boolean function depend on the values contained in its binary weight transform for the zero-valued vector.

A new probabilistic function of a Boolean function is introduced in Sect. 3. This function estimates the probabilistic distribution of bits at the output of a Boolean function if the distribution of the arguments, the function depends on, is known. Further, we suggest a characteristic for a balanced Boolean function that measures its ability to compensate a nonuniform distribution of the input. Resilient functions [5] are proved to have good compensating qualities.

2 Tensor Transform of Boolean Functions

Let $M_n(P)$ denote the ring of n-dimensional square matrices over the field P. For a pair of matrices $A \in M_n(P)$ and $B \in M_m(P)$ let $A \otimes B$ denote the Kronecker product [6, p. 421] of these matrices and $A^{[k]}$ denote the kth Kronecker power of A. For any matrix $A \in M_{2^n}(P)$ by writing $A = (\boldsymbol{g}_0, \ldots, \boldsymbol{g}_{2^n-1})$ we mean that $\boldsymbol{g}_i$ $(i = 0, \ldots, 2^n - 1)$ is the ith column of A, entries in $\boldsymbol{g}_i$ are indexed lexicographically by the elements in $\{0,1\}^n$, so

$$\boldsymbol{g}_i = \begin{pmatrix} g_i(0,\ldots,0) \\ g_i(0,\ldots,1) \\ \vdots \\ g_i(1,\ldots,1) \end{pmatrix} .$$

Let α_i $(i = 0, \ldots, 2^n - 1)$ denote the n-bit binary expansion of i, so $\boldsymbol{g}_i = (g_i(\alpha_0), \ldots, g_i(\alpha_{2^n-1}))^T$, where the superscript T denotes transpose of a matrix.

Lemma 1. *Let $A = (\boldsymbol{g}_0, \ldots, \boldsymbol{g}_{2^n-1}) \in M_{2^n}(P)$ and $A' = (\boldsymbol{g}'_0, \ldots, \boldsymbol{g}'_{2^{n-1}-1}) \in M_{2^{n-1}}(P)$. Suppose that $A = B \otimes A'$ for some matrix $B = \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$. Then*

$$g_i(x_1, \ldots, x_n) = \begin{cases} (b_{00}\overline{x_1} + b_{10}x_1)g'_{i'}(x_2, \ldots, x_n), & \text{if} \quad \alpha_i = (0, \alpha_{i'}), \\ (b_{01}\overline{x_1} + b_{11}x_1)g'_{i'}(x_2, \ldots, x_n), & \text{if} \quad \alpha_i = (1, \alpha_{i'}) \end{cases} ,$$

where $\alpha_{i'}$ is the $(n-1)$-bit vector, binary expansion of i'.

Proof. By the definition of the Kronecker product, $A = \begin{pmatrix} b_{00}A' & b_{01}A' \\ b_{10}A' & b_{11}A' \end{pmatrix}$. Thus,

$$g_i(0, x_2, \ldots, x_n) = \begin{cases} b_{00}g'_{i'}(x_2, \ldots, x_n), & \text{if} \quad \alpha_i = (0, \alpha_{i'}), \\ b_{01}g'_{i'}(x_2, \ldots, x_n), & \text{if} \quad \alpha_i = (1, \alpha_{i'}) \end{cases} \quad \text{and}$$

$$g_i(1, x_2, \ldots, x_n) = \begin{cases} b_{10} g'_{i'}(x_2, \ldots, x_n), & \text{if} \quad \alpha_i = (0, \alpha_{i'}), \\ b_{11} g'_{i'}(x_2, \ldots, x_n), & \text{if} \quad \alpha_i = (1, \alpha_{i'}) \end{cases} .$$

These equations combined together prove the claimed result. □

The following proposition easily follows from Lemma 1.

Proposition 1. *Let $A = B_1 \otimes \ldots \otimes B_n$, where $B_j = \begin{pmatrix} b_{00}^{(j)} & b_{01}^{(j)} \\ b_{10}^{(j)} & b_{11}^{(j)} \end{pmatrix}$ for $j = 1, \ldots, n$, and $A = (\boldsymbol{g}_0, \ldots, \boldsymbol{g}_{2^n-1})$. Then for any $i \in \{0, \ldots, 2^n - 1\}$*

$$g_i(x_1, \ldots, x_n) = \prod_{j=1}^{n} \left(\overline{\alpha_i^j} \left(b_{00}^{(j)} \overline{x_j} + b_{10}^{(j)} x_j \right) + \alpha_i^j \left(b_{01}^{(j)} \overline{x_j} + b_{11}^{(j)} x_j \right) \right) ,$$

where $\alpha_i = (\alpha_i^1, \ldots, \alpha_i^n)$.

Let $A \in M_{2^n}(P)$ be an invertible matrix and $A = (\boldsymbol{g}_0, \ldots, \boldsymbol{g}_{2^n-1})$. Further, let the function $f(x_1, \ldots, x_n)$, mapping $\{0,1\}^n$ in P, be defined by its string of values $T^f = (f(\alpha_0), \ldots, f(\alpha_{2^n-1}))^T \in P^{2^n}$ and let the function $F(x_1, \ldots, x_n)$ be defined by the string $T^F = A^{-1} T^f = (F(\alpha_0), \ldots, F(\alpha_{2^n-1}))^T \in P^{2^n}$. Vectors T^f and T^F are considered further as column-vectors. Then $T^f = AT^F$,

$$T^f = \sum_{i=0}^{2^n-1} \boldsymbol{g}_i F(\alpha_i) \quad \text{and} \quad f(x_1, \ldots, x_n) = \sum_{i=0}^{2^n-1} g_i(x_1, \ldots, x_n) F(\alpha_i) \qquad (1)$$

for any $(x_1, \ldots, x_n) \in \{0,1\}^n$. Equations (1) represent the decomposition of function f in the basis vector set $(\boldsymbol{g}_0, \ldots, \boldsymbol{g}_{2^n-1})$. Hereafter in this paper, by $f_{i_1, \ldots, i_m}^{\beta_1, \ldots, \beta_m}$ for any $1 \leq i_1 < \ldots < i_m \leq n$, we denote the subfunction of f obtained by fixing the variables $x_{i_1}, \ldots, x_{i_m}$ with binary values $\beta_1, \ldots, \beta_m$ respectively.

It is well known that if B_1 and B_2 are invertible matrices over P then the Kronecker product matrix $B_1 \otimes B_2$ is invertible too and $(B_1 \otimes B_2)^{-1} = B_1^{-1} \otimes B_2^{-1}$. In particular, if $B \in M_2(P)$ is an invertible matrix and $A = B^{[n]}$ then A is invertible too and $A^{-1} = (B^{-1})^{[n]}$.

Now we will demonstrate how Proposition 1 substantially facilitates proving of some important matrix identities for various representations of a function of Boolean variables. By convention, for a Boolean variable x we assume that $x^0 = \overline{x}$ and $x^1 = x$.

The Algebraic Normal Transform. Take $P = \mathrm{GF}(2)$ and set $B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = B^{-1}$ and $A = B^{[n]}$. Then, by Proposition 1,

$$g_i(x_1, \ldots, x_n) = \prod_{j=1}^{n} \left(\overline{\alpha_i^j} + \alpha_i^j x_j \right) = \prod_{j=1,\ldots,n \colon \alpha_i^j = 1} x_j \qquad (2)$$

and

$$f(x_1,\dots,x_n) \overset{(1)}{=} \sum_{i=0}^{2^n-1} g_i(x_1,\dots,x_n)F(\alpha_i) = \sum_{i=0}^{2^n-1} \left(\prod_{j=1,\dots,n:\, \alpha_i^j=1} x_j \right) F(\alpha_i) \ .$$

One can easily recognize the ANF of function f on the right hand side of the last identity, where $F(\alpha_i)$ $(i = 0,\dots,2^n-1)$ are the coefficients of the ANF polynomial. Let P^f denote the coefficient vector of the ANF polynomial for function f and denote also $R_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $R_{2^n} = R_2^{[n]}$. Then

$$T^f = R_{2^n} P^f \quad \text{and} \quad P^f = R_{2^n} T^f \ . \tag{3}$$

This transform of f is called the algebraic normal transform.

If R_{2^n} is considered as a matrix over the real number field $\mathbb{R}$ and the algebraic normal transform of f is implemented over $\mathbb{R}$ then T^F is equal to the coefficient vector of a real-valued, square-free (in variables) polynomial of n variables with integer coefficients that takes on the same values as function f on the points from GF$(2)^n$. Let Π_f denote the coefficient vector of such a polynomial. In this case $R_2^{-1} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$, $R_{2^n}^{-1} = (R_2^{-1})^{[n]}$,

$$T^f = R_{2^n} \Pi^f \quad \text{and} \quad \Pi^f = R_{2^n}^{-1} T^f \ . \tag{4}$$

This real-valued polynomial gives an important insight in certain probabilistic properties of a Boolean function that will be discussed further in Sect. 3.

The Probabilistic Transform. Assume that $P = \mathbb{R}$ and set $B = \frac{1}{2}\begin{pmatrix} 2 & -1 \\ 2 & 1 \end{pmatrix}$ and $A = B^{[n]}$. Then $B^{-1} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ -2 & 2 \end{pmatrix}$ and, by Proposition 1,

$$\begin{aligned} g_i(x_1,\dots,x_n) &= \prod_{j=1}^{n} \left(\overline{\alpha_i^j} + \frac{1}{2}\alpha_i^j (x_j - \overline{x_j}) \right) = \\ &= \prod_{j=1,\dots,n:\, \alpha_i^j=1} \frac{1}{2}(x_j - \overline{x_j}) \overset{(\circ)}{=} \prod_{j=1,\dots,n:\, \alpha_i^j=1} \delta_j \ , \end{aligned}$$

where $(\circ)$ is obtained by using $\overline{x_j} = 1 - x_j$ and introducing the new variable $\delta_j := x_j - 1/2$. Therefore,

$$f(x_1,\dots,x_n) \overset{(1)}{=} \sum_{i=0}^{2^n-1} g_i(x_1,\dots,x_n)F(\alpha_i) = \sum_{i=0}^{2^n-1} \left(\prod_{j=1,\dots,n:\, \alpha_i^j=1} \delta_j \right) F(\alpha_i) \ .$$

The right hand side of the last identity contains the real-valued, square-free polynomial of n variables $\delta_1,\dots,\delta_n$ that for $\{\delta_1,\dots,\delta_n\} \in \{-1/2, 1/2\}^n$ takes on the same values as function f on corresponding arguments $\{x_1,\dots,x_n\}$

if identity $x_j = \delta_j + 1/2$ is assumed. Therefore, if $D_f(x_1,\ldots,x_n)$ denotes a polynomial obtained by the algebraic normal transform over the reals then the probabilistic transform gives coefficients for polynomial $D_f(1/2+\delta_1,\ldots,1/2+\delta_n)$ that we will denote by Δ^f. Denote also $Q_2 = \frac{1}{2}\begin{pmatrix} 2 & -1 \\ 2 & 1 \end{pmatrix}$, $Q_{2^n} = Q_2^{[n]}$. Then $Q_2^{-1} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ -2 & 2 \end{pmatrix}$, $Q_{2^n}^{-1} = (Q_2^{-1})^{[n]}$,

$$T^f = Q_{2^n}\Delta^f \quad \text{and} \quad \Delta^f = Q_{2^n}^{-1}T^f \ . \tag{5}$$

We will call this transform of f the *probabilistic transform.* Applications of this transform will be discussed further in Sect. 3.

The Walsh Transform. According to [1, p. 118], the direct and inverse *Walsh transforms* of a real-valued function f over GF$(2)^n$ are defined as

$$S_f(\alpha_i) = \sum_{\boldsymbol{x}=0}^{2^n-1} f(\boldsymbol{x})(-1)^{\langle \alpha_i,\boldsymbol{x}\rangle} \quad \text{and} \quad f(\boldsymbol{x}) = \frac{1}{2^n}\sum_{i=0}^{2^n-1} S_f(\alpha_i)(-1)^{\langle \alpha_i,\boldsymbol{x}\rangle} \ , \tag{6}$$

where $\boldsymbol{x} = (x_1,\ldots,x_n)$ and $\langle \alpha_i,\boldsymbol{x}\rangle = \alpha_i^1 x_1 \oplus \ldots \oplus \alpha_i^n x_n$ is the standard inner product over GF(2). In the sum over $\boldsymbol{x}$ in (6) the summation index is considered as an integer in the range $0,\ldots,2^n-1$ but written in its binary expansion. The vector $S^f = (S_f(\alpha_0),\ldots,S_f(\alpha_{2^n-1}))$ is called the Walsh transform of function f.

Assume that $P = \mathbb{R}$ and set $B = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = 2B^{-1}$ and $A = B^{[n]}$. Thus, A is a Hadamard matrix of order 2^n (see [6, p. 422]). Then, by Proposition 1,

$$g_i(x_1,\ldots,x_n) = \prod_{j=1}^{n}\left(\overline{\alpha_i^j} + \alpha_i^j(\overline{x_j} - x_j)\right) = \prod_{j=1,\ldots,n:\,\alpha_i^j=1}(\overline{x_j} - x_j) = (-1)^{\langle \alpha_i,\boldsymbol{x}\rangle}$$

and

$$f(x_1,\ldots,x_n) \overset{(1)}{=} \sum_{i=0}^{2^n-1} g_i(x_1,\ldots,x_n)F(\alpha_i) = \sum_{i=0}^{2^n-1} F(\alpha_i)(-1)^{\langle \alpha_i,\boldsymbol{x}\rangle} \ .$$

In the latest identity one can recognize the inverse Walsh transform (6) but without the multiplicative coefficient. Therefore, in this case $F(\alpha_i) = 1/2^n S_f(\alpha_i)$, where $S_f(\alpha_i)$ is the Walsh transform of f evaluated in α_i. Let $H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ and $H_{2^n} = H_2^{[n]}$. Then

$$T^f = \frac{1}{2^n}H_{2^n}S^f \quad \text{and} \quad S^f = H_{2^n}T^f \ . \tag{7}$$

It is possible to generalize property (7) of the Walsh transform. Let us assume that function f is Boolean. From now on $wt(\omega)$ denotes the Hamming weight of a binary string ω and $wt(f)$ denotes the Hamming weight of a Boolean function f, i.e. the weight of T^f. Let r be an integer in the range $1 \le r \le n$ and let $i_1, \ldots, i_r$ be a set of indices with $1 \le i_1 < \ldots < i_r \le n$. Let $k_1, \ldots, k_{n-r}$ with $1 \le k_1 < \ldots < k_{n-r} \le n$ denote the indices complementing $i_1, \ldots, i_r$ with respect to $\{1, \ldots, n\}$. Let also the real-valued function $w(y_1, \ldots, y_r)$ of r Boolean variables be defined as follows

$$w(\alpha_j^1, \ldots, \alpha_j^r) = wt\left(f_{i_1,\ldots,i_r}^{\alpha_j^1,\ldots,\alpha_j^r}(x_{k_1}, \ldots, x_{k_{n-r}}) \right) = w_j$$

for $0 \le j < 2^r$, where $(\alpha_j^1, \ldots, \alpha_j^r) = \alpha_j$ is the r-bit binary expansion of j. Then, by (7), $S^w = H_{2^r}(w_0, \ldots, w_{2^r-1})^T$. On the other hand,

$$S_w(\alpha_i) \overset{(6)}{=} \sum_{j=0}^{2^r-1} w(\alpha_j)(-1)^{\langle \alpha_j, \alpha_i \rangle} = \sum_{j=0}^{2^r-1} \sum_{t=0}^{2^{n-r}-1} f_{i_1,\ldots,i_r}^{\alpha_j^1,\ldots,\alpha_j^r}(\alpha_t)(-1)^{\langle \alpha_j, \alpha_i \rangle} =$$
$$= \sum_{k=0}^{2^n-1} f(\alpha_k)(-1)^{\langle \alpha_k, \theta_i \rangle} \overset{(6)}{=} S_f(\theta_i) \ ,$$

where θ_i is the n-bit vector whose coordinates at the index positions $i_1, \ldots, i_r$ are equal to $\alpha_i^1, \ldots, \alpha_i^r$ respectively (where $(\alpha_i^1, \ldots, \alpha_i^r) = \alpha_i$) and the remaining $(n-r)$ coordinates are set to zero. Thus,

$$H_{2^r}(w_0, \ldots, w_{2^r-1})^T = (S_f(\theta_0), \ldots, S_f(\theta_{2^r-1}))^T \ , \tag{8}$$

which is the generalization of [5, Proposition 3.1], while the proof here is less complicated. If r is set equal to n then $w_j = f(\alpha_j)$, $\theta_i = \alpha_i$ and (8) transforms into (7).

If function f is Boolean then in some cases it is more convenient to work with the real-valued counterpart (sign function) of f, defined as $\hat{f}(\boldsymbol{x}) = 1 - 2f(\boldsymbol{x})$, and to apply the Walsh transform to $\hat{f}$. Function $\hat{f}$ can be recovered by the inverse Walsh transform of $S^{\hat{f}}$. Further, since $f(\boldsymbol{x}) = 1/2 - 1/2\hat{f}(\boldsymbol{x})$, the original function f can be obtained from the Walsh transform $S^{\hat{f}}$ by the following inverse transform:

$$f(\boldsymbol{x}) = \frac{1}{2} - \frac{1}{2^{n+1}} \sum_{i=0}^{2^n-1} S_{\hat{f}}(\alpha_i)(-1)^{\langle \alpha_i, \boldsymbol{x} \rangle} \ .$$

The relationship between the Walsh transform of $f(\boldsymbol{x})$ and $\hat{f}(\boldsymbol{x})$ is given by [7, Lemma 1] as follows

$$S_{\hat{f}}(0) = 2^n - 2S_f(0) \quad \text{and} \quad S_{\hat{f}}(w) = -2S_f(w) \quad \text{for} \quad 0 < w < 2^n \ . \tag{9}$$

By these identities and (7),

$$T^f = \left(\frac{1}{2}, \ldots, \frac{1}{2}\right)^T - \frac{1}{2^{n+1}} H_{2^n} S^{\hat{f}} \quad \text{and} \quad S^{\hat{f}} = (2^n, 0, \ldots, 0)^T - 2H_{2^n} T^f \tag{10}$$

since $H_{2^n}\left(\frac{1}{2},0,\ldots,0\right)^T=\left(\frac{1}{2},\ldots,\frac{1}{2}\right)^T$. On the other hand, identities, similar to (7), hold:

$$T^{\hat{f}}=\frac{1}{2^n}H_{2^n}S^{\hat{f}} \quad \text{and} \quad S^{\hat{f}}=H_{2^n}T^{\hat{f}} \ .$$

Combining (3) with (7) or (10), we obtain the following identities relating the coefficient vector of the ANF polynomial of f with the Walsh transforms S^f and $S^{\hat{f}}$:

$$P^f=\frac{1}{2^n}R_{2^n}H_{2^n}S^f \pmod 2=\frac{1}{2^n}\begin{pmatrix}1\,1\\2\,0\end{pmatrix}^{[n]}S^f \pmod 2 \qquad (11)$$

$$P^f=R_{2^n}\left(\left(\frac{1}{2},\ldots,\frac{1}{2}\right)^T-\frac{1}{2^{n+1}}H_{2^n}S^{\hat{f}}\right) \pmod 2 \ ,$$

where all operations on the right hand side are performed in $\mathbb{R}$ and the final result is reduced modulo 2.

Finally, if (5) is combined with (7) then the resulting identities relate the probabilistic transform of f with the Walsh transform S^f:

$$\Delta^f=\frac{1}{2^n}\begin{pmatrix}1&0\\0&-2\end{pmatrix}^{[n]}S^f \quad \text{and} \quad S^f=\begin{pmatrix}2&0\\0&-1\end{pmatrix}^{[n]}\Delta^f \ . \qquad (12)$$

Since the matrix of the transform (12) is diagonal, coordinates of zero values in vectors Δ^f and S^f are the same. Now, using Proposition 1 for $B=\begin{pmatrix}1&0\\0&-2\end{pmatrix}$ and $A=B^{[n]}$, we obtain

$$g_i(x_1,\ldots,x_n)=\prod_{j=1}^{n}\left(\overline{\alpha_i^j}\,\overline{x_j}-2\alpha_i^j x_j\right)=\begin{cases}(-2)^{wt(\alpha_i)}, & \text{if}\, x_j=\alpha_i^j\ (j=1,\ldots,n),\\ 0, & \text{otherwise}\end{cases} .$$

Therefore, by (12),

$$\Delta_f(\omega)=\frac{1}{2^n}(-2)^{wt(\omega)}S_f(\omega)\overset{(9)}{=}\begin{cases}\frac{1}{2^n}(-2)^{wt(\omega)-1}S_{\hat{f}}(\omega), & \text{if} \quad \omega\neq 0\\ \frac{1}{2}-\frac{1}{2^{n+1}}S_{\hat{f}}(0), & \text{if} \quad \omega=0\end{cases} , \qquad (13)$$

where $\Delta_f(\omega)$ is the ωth coordinate of the probabilistic transform of function f.

The Weight Transform. Take $P=\mathbb{R}$ and set $B_0=\begin{pmatrix}0&1\\1&-1\end{pmatrix}$, $B_1=\begin{pmatrix}1&-1\\0&1\end{pmatrix}$ and $A=B_{\beta_1}\otimes\ldots\otimes B_{\beta_n}$ for some n-bit vector $\beta=(\beta_1,\ldots,\beta_n)$. Let also $A^{-1}=B_{\beta_1}^{-1}\otimes\ldots\otimes B_{\beta_n}^{-1}=(\tilde{\boldsymbol{g}}_0,\ldots,\tilde{\boldsymbol{g}}_{2^n-1})$, where $B_0^{-1}=\begin{pmatrix}1\,1\\1\,0\end{pmatrix}$ and $B_1^{-1}=\begin{pmatrix}1\,1\\0\,1\end{pmatrix}$. By Proposition 1 and since $b_{11}^{(j)}=b_{12}^{(j)}=1$ for any $j=1,\ldots,n$,

$$\tilde{g}_i(x_1,\ldots,x_n)=\prod_{j=1}^{n}\left(\overline{x_j}+x_j\left(\overline{\alpha_i^j}b_{21}^{(j)}+\alpha_i^j b_{22}^{(j)}\right)\right)\overset{(*)}{=}$$

$$\overset{(*)}{=}\prod_{j=1}^{n}\left(\overline{x_j}+x_j\left(\overline{\alpha_i^j}\,\overline{\beta_j}+\alpha_i^j\beta_j\right)\right)=\prod_{j=1,\ldots,n:\,\alpha_i^j\neq\beta_j}\overline{x_j} \ . \qquad (14)$$

Equality (*) holds because $b_{21}^{(j)} = \overline{\beta_j}$ and $b_{22}^{(j)} = \beta_j$. Thus, $\tilde{g}_i(x_1, \dots, x_n)$ is equal to one if and only if the coordinates, where vectors α_i and β differ, correspond to the zero entries in vector $(x_1, \dots, x_n)$.

Let us assume that function f is Boolean. Then

$$F(x_1, \dots, x_n) = \sum_{i=0}^{2^n-1} \tilde{g}_i(x_1, \dots, x_n) f(\alpha_i) =$$
$$= \sum_{i=0}^{2^n-1} \left(\prod_{j=1,\dots,n:\, \alpha_i^j \neq \beta_j} \overline{x_j} \right) f(\alpha_i) = wt\left(f_{t_1,\dots,t_k}^{\beta_{t_1},\dots,\beta_{t_k}} \right) ,$$

where $k = wt(x_1, \dots, x_n)$ and $t_1, \dots, t_k$ are the coordinates of the nonzero entries in $(x_1, \dots, x_n)$. Here it is assumed that if $\alpha_i = \beta$ then $\prod_{j=1,\dots,n:\, \alpha_i^j \neq \beta_j} \overline{x_j} = 1$. Therefore, $wt\left(f_{1,\dots,n}^{\beta_1,\dots,\beta_n} \right) = f(\beta)$.

Let Θ_β^f denote the ordered 2^n-tuple, containing the weights of the subfunctions of f, obtained by fixing all possible subsets of variables with corresponding values from vector β. Thus,

$$\Theta_\beta^f = \left\{ wt\left(f_{i_1,\dots,i_k}^{\beta_{i_1},\dots,\beta_{i_k}} \right) \mid 1 \le i_1 < \dots < i_k \le n;\ k \in \{0, \dots, n\} \right\} .$$

Denote also $D_\beta = B_{\beta_1} \otimes \dots \otimes B_{\beta_n}$. Then

$$T^f = D_\beta \, \Theta_\beta^f \quad \text{and} \quad \Theta_\beta^f = D_\beta^{-1} T^f \ . \tag{15}$$

We will call this transform of f the *weight transform*. In particular, if vector β consists of zeros only then $D_\beta = B_0^{[n]}$, and if it consists only of ones then $D_\beta = B_1^{[n]}$.

If we consider matrices B_0 and B_1 as matrices over the field GF(2) and perform all operations in (15) in this field then (15) will relate the string of values of function f with binary weights of its subfunctions.

Let us compare the basis vector set (14) of the weight transform when $\beta = (0, \dots, 0)$ with the basis vector set (2) of the inverse algebraic normal transform. It is clear that they are directly related via a simple variable complementation. Since $R_{2^n} = R_{2^n}^{-1}$, the basis vector sets of the algebraic normal transform and its inverse are equal. Therefore,

$$P_f(\alpha_i^1, \dots, \alpha_i^n) = \Theta_0^f(\overline{\alpha_i^1}, \dots, \overline{\alpha_i^n}) \ (\text{mod } 2) \tag{16}$$

for any $i = 0, \dots, 2^n - 1$, where $(\alpha_i^1, \dots, \alpha_i^n) = \alpha_i$. This identity is easily accounted for by the well-known fact that a Boolean function has maximal algebraic degree if and only if it has an odd weight. Indeed, the right hand side of the identity contains the binary weight of the subfunction which maximal possible order term in the ANF is equal to $\prod_{j=1,\dots,n:\, \alpha_i^j = 1} x_j$ and the coefficient for this term in the ANF of f is the value on the left hand side of the identity. To construct the subfunction, relevant variables of f are being fixed only with zero values, therefore, the term $\prod_{j=1,\ldots,n:\,\alpha_i^j=1} x_j$ is either present in the ANFs of both f and the subfunction or is missing in both.

The weight transform defines correlation properties of a Boolean function and these properties are extremely important, especially for stream cipher design, where filter and combination generators with not correlation immune filtering and combining functions are susceptible to ciphertext-only attacks [2]. Therefore, the weight transform is important for assessing cryptographic characteristics of Boolean functions.

It is important to note that the P^f, Π^f, Δ^f, S^f, $S^{\hat{f}}$ and Θ_β^f transforms of a function f can be represented by matrix equations (3), (4), (5), (7), (10) and (15), all based on the Kronecker product of appropriate elementary cells. This fact allows to use fast Fourier and Walsh transform algorithms [8] for efficient estimation of these transforms and easy transition from one transform to another.

3 Probabilistic Function of a Boolean Function

Let us consider the arrangement when n sequences of nonuniform, independent and identically distributed (i.i.d.) random binary variables are combined with a Boolean function to produce an output sequence hopefully having better algebraic and statistical properties relevant to a key-stream. In this section we show that an appropriately chosen combining function can compensate the nonuniform distribution of the inputs and generate the close-to-uniform output. Similar problems were considered in a recent paper [9] where maximized estimates for the bias of the distribution of the output bits were made. Our approach allows to obtain explicit polynomial expression for this bias.

Definition 1. *Let $f(x_1,\ldots,x_n)$ be a Boolean function of n variables. Assume that $\boldsymbol{X}=(X_1,\ldots,X_n)$ is an n-tuple consisting of i.i.d. random binary variables with $P(X_i=1)=p_i$ for $i=1,\ldots,n$. Then function $F_f(p_1,\ldots,p_n)=P(f(\boldsymbol{X})=1)$ is called the probabilistic function of f.*

From Definition 1 it follows that $F_f(p_1,\ldots,p_n)=\sum_{\beta:\,f(\beta)=1} P(\boldsymbol{X}=\beta)$ and if $\beta=(\beta_1,\ldots,\beta_n)$ then $P(\boldsymbol{X}=\beta)=\prod_{i=1}^n p_i^{\beta_i}(1-p_i)^{1-\beta_i}$. Thus, $F_f(p_1,\ldots,p_n)$ is the polynomial of n variables $p_1,\ldots,p_n$ with integer coefficients.

Further, let $D_f(x_1,\ldots,x_n)$ denote the real-valued, square-free (in variables) polynomial of n variables with integer coefficients such that

$$D_f(x_1,\ldots,x_n)=f(x_1,\ldots,x_n) \quad \text{for any} \quad (x_1,\ldots,x_n)\in \mathrm{GF}(2)^n \ . \tag{17}$$

Let us write down the polynomial D_f in the canonical form

$$D_f(x_1,\ldots,x_n)=\sum_{i=0}^{2^n-1} a_i \left(\prod_{j=1,\ldots,n:\,\alpha_i^j=1} x_j\right) ,$$

where $\alpha_i = (\alpha_i^1, \ldots, \alpha_i^n)$ is the n-bit binary expansion of i and $a_i \in \mathbb{Z}$. Then, since (17) holds, the integer coefficients a_i form the solution of the following system of linear equations

$$M(a_0, \ldots, a_{2^n-1})^T = (f(0, \ldots, 0), \ldots, f(1, \ldots, 1))^T \ ,$$

where $M = (m_{i,j})_{2^n \times 2^n}$ $(i, j = 0, \ldots, 2^n - 1)$ is a nondegenerate triangular $\{0, 1\}$-matrix with $m_{i,j} = 1$ if and only if the positions of ones in the n-bit binary expansion of j are a subset of those in the binary expansion of i (in particular, it is necessary that $j \le i$). Therefore, this system has a unique solution and that proves the *uniqueness* of the polynomial D_f. Moreover, the coefficient vector of D_f can be obtained by the algebraic normal transform of function f over $\mathbb{R}$ (see Sect. 2).

Identities $\overline{x} = 1 - x$, $x_1 \wedge x_2 = x_1 x_2$ and $x_1 \oplus x_2 = x_1 + x_2 - 2x_1 x_2$ convert elementary Boolean operations into integer expressions. Thus, using these identities any formula representing $f(x_1, \ldots, x_n)$ in the basis $\{^-, \wedge, \oplus\}$ (for instance, the ANF) can be transformed into the real-valued polynomial of n variables with integer coefficients that satisfies (17). Moreover, if we assume that $x_i^2 \equiv x_i$ $(i = 1, \ldots, n)$ then the constructed polynomial is square-free and, therefore, by the uniqueness, is equal to D_f. That provides an alternative way for constructing polynomial D_f starting from a formula representing the Boolean function.

Proposition 2. *For any Boolean function $f(x_1, \ldots, x_n)$ and arbitrary values $p_1, \ldots, p_n$ with $0 \le p_i \le 1$ for all $i = 1, \ldots, n$*

$$F_f(p_1, \ldots, p_n) = D_f(p_1, \ldots, p_n) \ .$$

Proof. To prove this identity we apply induction on n.

Let $n = 1$. Then function f is one of the following four functions of a single variable

$$f_0 \equiv 0, \quad f_1 = x_1, \quad f_2 = \overline{x_1}, \quad f_3 \equiv 1 \ .$$

But

$$\begin{aligned} &P(f_0 = 1) = 0 = D_{f_0} \\ &P(f_1 = 1) = P(X_1 = 1) = p_1 = D_{f_1}(p_1) \\ &P(f_2 = 1) = P(X_1 = 0) = 1 - p_1 = D_{f_2}(p_1) \\ &P(f_3 = 1) = 1 = D_{f_3} \ . \end{aligned}$$

Now, supposing that the proposition is true for $n = l - 1$, we prove it for $n = l$. It is easy to see that the following decomposition of function f into subfunctions holds:

$$f(x_1, \ldots, x_l) = \overline{x_1} f_1^0(x_2, \ldots, x_l) \oplus x_1 f_1^1(x_2, \ldots, x_l) \ .$$

According to the induction hypothesis, $F_{f_1^i}(p_2, \ldots, p_l) = D_{f_1^i}(p_2, \ldots, p_l)$ for $i = 0, 1$. On the other hand,

$$D_f(x_1, \ldots, x_l) = (1 - x_1) D_{f_1^0}(x_2, \ldots, x_l) + x_1 D_{f_1^1}(x_2, \ldots, x_l)$$

since $\overline{x_1}f_1^0(x_2,\dots,x_l)x_1f_1^1(x_2,\dots,x_l) \equiv 0$ on $\mathrm{GF}(2)^n$. On the other hand, by the rule of total probability

$$F_f(p_1,\dots,p_l) = (1-p_1)F_{f_1^0}(p_2,\dots,p_l) + p_1F_{f_1^1}(p_2,\dots,p_l) \ .$$

Thus, $F_f(p_1,\dots,p_l) = D_f(p_1,\dots,p_l)$ for any $p_1,\dots,p_n$ with $0 \le p_i \le 1$ for all $i = 1,\dots,n$. □

Let w_i $(i = 0,\dots,n)$ denote the number of vectors having the weight i in the support of a Boolean function f of n variables. Let us assume first that $p_1 = \dots = p_n = p = 1/2+\delta$, where $\delta \in (-1/2,1/2)$ is the bias of the distribution of the random variable x_i $(i = 1,\dots,n)$. Then, since $\sum_{i=0}^n w_i = wt(f)$,

$$F_f(p) = \sum_{i=0}^n w_ip^i(1-p)^{n-i} = \sum_{i=0}^n w_i\left(\frac{1}{2}+\delta\right)^i\left(\frac{1}{2}-\delta\right)^{n-i} =$$
$$= d_1\delta + d_2\delta^2 + \dots + d_n\delta^n + \frac{1}{2^n}wt(f) \ ,$$

where $d_1,\dots,d_n$ are some real values. Let $\Delta_f(\delta) = F_f(1/2+\delta) - 1/2$ denote the bias of the distribution of the function f output. In particular, if function f is balanced then $\Delta_f(\delta) = d_1\delta + d_2\delta^2 + \dots + d_n\delta^n$.

In case when the values of $p_1,\dots,p_n$ are different let $p_i = 1/2 + \delta_i$ $(i = 1,\dots,n)$. The bias of the distribution of the function f output is defined in a similar way as the polynomial of n variables

$$\Delta_f(\delta_1,\dots,\delta_n) = F_f\left(\frac{1}{2}+\delta_1,\dots,\frac{1}{2}+\delta_n\right) - \frac{1}{2} \ . \tag{18}$$

If function f is balanced then the constant term of polynomial $\Delta_f(\delta_1,\dots,\delta_n)$ is equal to

$$\Delta_f(0,\dots,0) = F_f\left(\frac{1}{2},\dots,\frac{1}{2}\right) - \frac{1}{2} = \frac{wt(f)}{2^n} - \frac{1}{2} = 0 \ .$$

And the other way around: if the constant term of polynomial $\Delta_f(\delta_1,\dots,\delta_n)$ is equal to zero then function f is balanced. We will call polynomial $\Delta_f(\delta_1,\dots,\delta_n)$ the *bias polynomial* of function f.

The coefficient vector of the bias polynomial is equal to the probabilistic transform of function f (see Sect. 2) except for the initial coordinate of Δ^f which has to be corrected by subtracting 1/2. On the other hand, combining (9) and (12), the coefficient vector can be expressed as $-\frac{1}{2^{n+1}}\begin{pmatrix}1 & 0\\ 0 & -2\end{pmatrix}^{[n]} S^{\hat{f}}$. Coefficients of the bias polynomial can also be estimated using identities (13) that are equivalent to [9, Theorem 3.1].

Definition 2. *For $k \in \{1,\dots,n\}$ a Boolean function f is called k-compensating if the bias polynomial of f does not contain product terms having degree lower than k.*

Note that any balanced Boolean function is 1-compensating. For the particular case when $p_1 = \ldots = p_n$, Definition 2 means that function f is k-compensating if it is balanced and $d_1 = \ldots = d_{k-1} = 0$. In other words, if the input of a k-compensating Boolean function is nonuniform with bias δ then the bias on its output is at most the size of order δ^k. The following proposition provides a method for constructing k-compensating functions.

Proposition 3. *Let $f(x_1, \ldots, x_n) = f_1(x_1, \ldots, x_k) \oplus f_2(x_{k+1}, \ldots, x_n)$, where $k \in \{1, \ldots, n-1\}$. Then*

$$F_f(p_1, \ldots, p_n) - \frac{1}{2} = -2\left(F_{f_1}(p_1, \ldots, p_k) - \frac{1}{2}\right)\left(F_{f_2}(p_{k+1}, \ldots, p_n) - \frac{1}{2}\right) ,$$

i.e., $\Delta_f(\delta_1, \ldots, \delta_n) = -2\Delta_{f_1}(\delta_1, \ldots, \delta_k)\Delta_{f_2}(\delta_{k+1}, \ldots, \delta_n)$.

Proof. Since $f_1 \oplus f_2 = f_1 + f_2 - 2f_1f_2$,

$$\begin{aligned} D_f(x_1, \ldots, x_n) = D_{f_1}(x_1, \ldots, x_k) + D_{f_2}(x_{k+1}, \ldots, x_n) - \\ - 2D_{f_1}(x_1, \ldots, x_k)D_{f_2}(x_{k+1}, \ldots, x_n) \ . \end{aligned}$$

Therefore, by Proposition 2,

$$\begin{aligned} F_f(p_1, \ldots, p_n) = F_{f_1}(p_1, \ldots, p_k) + F_{f_2}(p_{k+1}, \ldots, p_n) - \\ - 2F_{f_1}(p_1, \ldots, p_k)F_{f_2}(p_{k+1}, \ldots, p_n) \ , \end{aligned}$$

which is equivalent to the statement of the proposition. □

The following corollary is obvious.

Corollary 1. *Let $f(x_1, \ldots, x_n)$ be a Boolean function of n variables. If*

(i) $f(x_1, \ldots, x_n) = f_1(x_1, \ldots, x_k) \oplus f_2(x_{k+1}, \ldots, x_n)$, function f_1 is k_1-compensating and function f_2 is k_2-compensating then function f is $(k_1 + k_2)$-compensating;

(ii) $f(x_1, \ldots, x_n) = x_{i_1} \oplus \ldots \oplus x_{i_k} \oplus a_0$ then

$$F_f(p_1, \ldots, p_n) - \frac{1}{2} = (-1)^{a_0}(-2)^{k-1}\delta_{i_1} \cdot \ldots \cdot \delta_{i_k} \ .$$

In other words, an affine function consisting of k linear terms is k-compensating.

The following proposition easily follows from (12) and the well-known criterion of correlation immunity in terms of the Walsh transform [4].

Proposition 4. *A Boolean function $f(x_1, \ldots, x_n)$ is k-resilient if and only if it is $(k+1)$-compensating.*

Therefore, highly resilient Boolean functions significantly increase the size of order for the bias of the distribution of the output bits compared to the bias of the input. On the other hand, due to Siegenthaler's inequality [10, Theorem 1], Proposition 4 means that Boolean functions with high algebraic degree have poor compensating properties and vice versa. This fact underlines again the need for optimizing algebraic degree with correlation and compensating properties when constructing secure Boolean functions.

4 Conclusion

The classical algebraic normal and Walsh transforms appear to be a special case of the tensor transform that also allows for the definition of new transforms, in particular, probabilistic and weight transforms. The new transforms are cryptographically important since they relate a Boolean function directly to its bias polynomial and to the weights of its subfunctions. A tensor transform is based on the Kronecker product of appropriate elementary cells. This fact allows to use fast Fourier and Walsh transform algorithms for efficient estimation of any tensor transform and easy transition from one transform to another.

The probabilistic function allows to estimate the probabilistic distribution of bits at the output of a Boolean function if the distribution of the arguments, the function depends on, is known. The probabilistic function is a polynomial which coefficients can be obtained by the algebraic normal transform of a Boolean function over $\mathbb{R}$. The newly introduced measure for a Boolean function to compensate a nonuniform distribution of its input bits was called the compensating degree. Compensating degree can be efficiently estimated by the probabilistic transform. Highly resilient Boolean functions significantly increase the size of order for the bias of the distribution of the output bits compared to the bias of the input.

Acknowledgement

We would like to thank anonymous referees for their comments and suggestions which helped to improve the paper.

References

1. Rueppel, R.A.: Analysis and Design of Stream Ciphers. Communications and Control Engineering Series. Springer-Verlag, Berlin (1986)
2. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Transactions on Computers **C-34** (1985) 81–85
3. Ding, C., Xiao, G.Z., Shan, W.: The Stability Theory of Stream Ciphers. Volume 561 of LNCS. Springer-Verlag, Berlin (1991)
4. Xiao, G.Z., Massey, J.L.: A spectral characterization of correlation-immune combining functions. IEEE Transactions on Information Theory **34** (1988) 569–571
5. Sarkar, P.: Spectral domain analysis of correlation immune and resilient Boolean functions. Cryptology ePrint Archive, Report 2000/049 (2000)
6. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. North-Holland, Amsterdam (1996) Ninth impression.
7. Forré, R.: The strict avalanche criterion: Spectral properties of Boolean functions and an extended definition. In Goldwasser, S., ed.: Advances in Cryptology - Crypto '88. Volume 403 of LNCS., Berlin, Springer-Verlag (1989) 450–468
8. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Amsterdam (1974)
9. Miranovich, K.: Spectral analysis of Boolean functions under non-uniformity of arguments. Cryptology ePrint Archive, Report 2002/021 (2002)
10. Siegenthaler, T.: Correlation-immunity of nonlinear combining functions for cryptographic applications. IEEE Transactions on Information Theory **IT-30** (1984) 776–780

Related-Cipher Attacks

Hongjun Wu

Laboratories for Information Technology
21 Heng Mui Keng Terrace
Singapore 119613
hongjun@lit.a-star.edu.sg

Abstract. We formally introduce the concept of related-cipher attack. In this paper, we consider the related ciphers as block ciphers with the same round function but with different round numbers. If their key schedules do not depend on the total round number, then related-cipher attack could be applied if the same key is used. We applied this attack to block cipher SQUARE and show that SQUARE is vulnerable to this attack. We also show that a new AES key schedule proposed at ACISP02 is weaker than the original one under this attack. We then classify the differential attacks into three categories: related-message attack (the original differential cryptanalysis), related-key attack and related-cipher attack. These attacks should be taken into consideration in cipher design.

1 Introduction

There have been a number of attacks on block ciphers. The most important two kinds of attacks are differential cryptanalysis [1] and linear cryptanalyis [11]. There are some variants or extensions of these two attacks such as the higher order differential cryptanalysis [7], truncated differential cryptanalysis [5], multiple linear approximations [9], non-linear approximations [6], partitioning cryptanalysis [4] and differential-linear cryptanalysis [8], etc. A common feature of these attacks is that both the cipher and the key are fixed. By analyzing some known (or chosen) plaintexts, information about the key could be revealed. The linear cryptanalysis can also be applied in the ciphertext only attack when there is sufficient redundancy in the plaintexts. All these attacks are very important in the design of ciphers. In [2], the related-key attack is introduced. For this attack, the cipher is fixed while the keys are related. This attack can be applied when some related keys and one common cipher are used to encrypt messages. Related key attack has important implication on the key schedule design of block ciphers.

In this paper, we introduce a new attack – related-cipher attack. For related-cipher attack, the key is fixed while the ciphers are related. It could be applied when someone uses the same key in related ciphers. In this paper, we consider the related ciphers as block ciphers with the same round function but different round number and their key schedules do not depend on the total round number. This attack can find the key easily when the difference between the round numbers is small. Related-cipher attack has important implication on the design of the key schedule of block ciphers with flexible round number.

This paper is organized as follows. The related-cipher attack is introduced in Section 2. Section 3 applies the related-cipher attack to some block ciphers with flexible round number. Section 4 suggests a way to resist related cipher attack by relating the key schedule to the total round number of the block cipher. Section 5 concludes this paper.

2 Related-Cipher Attack

Usually a secret key is associated with one particular cipher. However, the same key may be used in different ciphers in some cases. If those ciphers are related, the related-cipher attack may be applied. In this paper, we deal with the block ciphers with flexible round number.

Flexible round number is a feature in some block ciphers. It allows a user to choose greater security level. However, the key schedule of some of these block ciphers does not depend on the total round number. We denote these block ciphers as **related ciphers**. For this kind of cipher, if the same key is used in the ciphers with different round number, then the key can be found when the difference between the round numbers is small. The attack is outlined below.

Related-Cipher Attack on Block Ciphers. Consider two related block ciphers. Both of them have the same round function, but with different round numbers, r and $(r + \Delta r)$ respectively. If a key is used in these two ciphers to encrypt the same message, the attack can be carried out on the Δr- round cipher. For this Δr-round cipher, the plaintext is the ciphertext of the r-round cipher and the ciphertext is that of the $(r + \Delta r)$-round cipher. The key can be determined easily for small Δr.

In the next section, we will apply related-cipher attack on some block ciphers with flexible round number.

3 Related-Cipher Attack on Some Block Ciphers

In this section related cipher attack is applied to two block ciphers with flexible round number. Block SQUARE [3] is vulnerable to this kind of attack. AES [13] can resist this kind of attack. But a new AES key schedule [12] is not that secure. Other block ciphers with flexible round number such as SAFER [10] are also vulnerable to the related cipher attack but we omit the attacks here. These results show that some of the block ciphers with flexible round number are really vulnerable to related cipher attack if their key schedules are not carefully designed. Care should be taken when we design block ciphers with flexible round number. In Subsection 3.1, we apply the attack to SQUARE. In Subsection 3.2, AES is shown to be able to resist the related-cipher attack. In Subsection 3.3, we show that a new AES key schedule presented at ACISP02 is not secure against the related-cipher attack.

3.1 Block Cipher SQUARE

SQUARE is a new block cipher designed by J. Daemen, L. Knudsen and V. Rijmen. The round number of SQUARE is set to eight while the designers also allow the conservative users to increase the number of rounds in a straight way. The key schedule of SQUARE does not depend on the total round number of the cipher and thus vulnerable to the related cipher attack.

3.1.1 Structure of SQUARE

The structure of SQUARE is outlined below. Interested readers may refer to [3] for the detail. SQUARE is an iterated block cipher with a block length and key length of 128 bits each. The basic building blocks of the cipher are five different invertible transformations that operate on a 4 × 4 array of bytes. The element of a state a in row i and column j is specified as $a_{i,j}$. Both indexes start from 0. These five transformations are outlined below.

A Linear Transformation θ.

$$\theta : b = \theta(a) \Leftrightarrow b_{i,j} = c_j a_{i,0} \oplus c_{j-1} a_{i,1} \oplus c_{j-2} a_{i,2} \oplus c_{j-3} a_{i,3}$$

where the multiplication in GF(2^8) and the indices of c is taken modulo 4. If the rows of a state is denoted by polynomials

$$a_i(x) = \bigoplus_{j=0}^{3} a_{i,j} x^j$$

Using this notation, and defining

$$c(x) = \bigoplus_{j=0}^{3} c_j x^j$$

Then θ can be described as a modular polynomial multiplication:

$$b = \theta(a) \Leftrightarrow b_i(x) = c(x) a_i(x) \bmod (1 + x^4) \text{ for } 0 \le i < 4$$

In SQUARE, $c(x)$ is chosen to be

$$c(x) = 2_x \oplus 1_x \cdot x \oplus 1_x \cdot x^2 \oplus 3_x x^3$$

A Nonlinear Transformation γ.
γ is a nonlinear byte substitution and is defined as

$$\gamma : b = \gamma(a) \Leftrightarrow b_{i,j} = S_\gamma(a_{i,j})$$

with S_γ an invertible 8-bit substitution table.

In SQUARE, the S-box is constructed by taking the mapping $x \to x^{-1}$ and applying an affine transformation to the output bits.

A Byte Permutation π.

$$\pi : b = \pi(a) \Leftrightarrow b_{i,j} = a_{j,i}$$

Bitwise Round Key Addition σ.

$\sigma[k^t]$ consists of the bitwise addition of a round key k^t.

$$\sigma[k^t] : b = \sigma[k^t](a) \Leftrightarrow b = a \oplus k^t$$

The Round Key Evolution ψ.
The round keys k^t are derived iteratively from the cipher key K in the following way.

$$\begin{aligned} k^0 &= K \\ k^t &= \psi(k^{t-1}) \end{aligned}$$

where ψ is an invertible affine transformation and $k^{t+1} = \psi(k^t)$ is defined by

$$\begin{aligned} k_0^{t+1} &= k_0^t \oplus rotl(k_3^t) \oplus C_t \\ k_1^{t+1} &= k_1^t \oplus k_0^{t+1} \\ k_2^{t+1} &= k_2^t \oplus k_1^{t+1} \\ k_3^{t+1} &= k_3^t \oplus k_2^{t+1} \end{aligned} \tag{1}$$

where $rotl(a_i)$ is a left byte-rotation operation on a row as

$$rotl[a_{i,0}a_{i,1}a_{i,2}a_{i,3}] = [a_{i,1}a_{i,2}a_{i,3}a_{i,0}]$$

and the round constants C_t are also defined iteratively as

$$\begin{aligned} C_0 &= 1_x \\ C_t &= 2_x \cdot C_{t-1} \end{aligned}$$

We notice that the key schedule of SQUARE does not depend on the total round number.

The Cipher SQUARE
The tth round function is denoted by $\rho[k^t]$:

$$\rho[k^t] = \sigma[k^t] \circ \pi \circ \gamma \circ \theta$$

SQUARE is defined as eight rounds preceeded by a key addition $\sigma[k^0]$ and by θ^{-1}:

$$\begin{aligned} \text{SQUARE}[k] = {} & \rho[k^8] \circ \rho[k^7] \circ \rho[k^6] \circ \rho[k^5] \circ \\ & \rho[k^4] \circ \rho[k^3] \circ \rho[k^2] \circ \rho[k^1] \circ \sigma[k^0] \circ \theta^{-1} \end{aligned}$$

As a safety margin, the designers fixed the number of rounds to eight. However, the designers also allow conservative users to increase the number of rounds in a straight way.

3.1.2 Related Cipher Attack on SQUARE

From the description of SQUARE, it is noted that the key schedule of SQUARE does not depend on the total round number. So SQAUARE ciphers with different round numbers are related. If the same key is used in SQUARE with different round number, then the related cipher attack can be applied.

Denote c^r as the ciphertext of r round SQUARE and $c^{r+\Delta r}$ as that of $r+\Delta r$ round SQUARE. If a c^r and a $c^{r+\Delta r}$ are related to the same plaintext, they are denoted as one right pair. We apply the related cipher attack to the situations where Δr =1 and Δr =2.

When $\Delta r = 1$, the cipher key can be determined from only one right pair. In this case, SQUARE is reduced to only one round and the following relation holds

$$c^{r+1} = \rho[k^{r+1}](c^r) \tag{2}$$

From equation (2), the round key k^{r+1} is derived as

$$k^{r+1} = (\pi \circ \gamma \circ \theta(c^r)) \oplus c^{r+1}$$

The cipher key K can be derived from this round key directly since the key evolution ψ is invertible. This cipher key K can be used to decrypt all the messages encrypted with it.

When $\Delta r = 2$, the cipher key can be determined from two right pairs easily. In this case, SQUARE is reduced to two rounds and the following relation holds

$$c^{r+2} = \rho[k^{r+2}] \circ \rho[k^{r+1}](c^r) \tag{3}$$

Let

$$c^{r'} = \pi \circ \gamma \circ \theta(c^r)$$

then equation (3) is simplified to

$$c^{r+2} = k^{r+2} \oplus (\pi \circ \gamma \circ \theta(c^{r'} \oplus k^{r+1})) \tag{4}$$

We note the fact that one row of $(c^{r'} \oplus k^{r+1})$ is related to one column of $(c^{r+2} \oplus k^{r+2})$. So equation (4) is decomposed into four block ciphers each with 32-bit block length. These four block ciphers have the common form as

$$c = k_2 \oplus (\gamma \circ \theta(c' \oplus k_1)) \tag{5}$$

The value of k_1 (or k_2) can be determined easily from 2 pairs of (c, c'). So the round key k^{r+1} (or k^{r+2}) is known and the cipher key can be determined.

3.2 Block Cipher AES

AES is also with flexible round number: 10 for AES-128, 12 for AES-192, and 14 for AES-256 (where AES-x indicates AES with x-bit secret key). However, AES is not vulnerable to the related-cipher attack.

3.2.1 Structure of AES

We introduce only the key schedule of AES here. Its pseudo code is given in Fig. 1.

In Fig. 1, the *key*[ ] represents the cipher key, *Nk* is the length of the cipher key in 32-bit words, *Nb* is the block size in words, *w*[ ] is the round keys, *Nr* is the round number. Subword, RotWord and Rcon are some functions we will omit their illustrations here.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word temp
   i = 0
   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1],
                  key[4*i+2], key[4*i+3])
      i = i+1
   end while
   i = Nk
   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Fig. 1. Pseudo code for AES key schedule

3.2.2 AES Is Able to Resist the Related-Cipher Attack

From the description of the AES key schedule, we see that the key schedule of AES depends on the key length *Nk*. It is thus impossible for the same key being used in the AES with different round numbers. Even if a 256-bit key is the repeat of a 128-bit key and both of them are used to encrypt the same message, the related-cipher attack could not be applied since the key schedule of AES-256 is slightly different from that of AES-128. We see that AES is able to resist the related cipher attack because the relationship between the key length and the round number is fixed. It is thus avoided that the same key being used in the related ciphers.

3.3 A New AES Key Schedule

At ACISP2002, a new AES key schedule [12] was proposed. However, we will show that this new key schedule is weaker than the original one under the related-cipher attack.

3.3.1 Description of the New AES Key Schedule

We introduce only the new key schedules for AES-192 and AES-256 here.

```
Let $p$ = 8,  $Nr$ = 12 for AES-192
    $p$ = 16, $Nr$ = 14 for AES-256
for $r$ = 0 to $Nr$
    for j = 0 to 15
        $a_j$ = $Mk_j \oplus S[r\times16+j] \oplus S[MK_{j+p}]$
        $a_j$ = $Mk_{j+p} \oplus S[r\times16+j] \oplus S[MK_j]$
    for $i$ = 0 to 2
        ByteSub($a$)
        ShiftRow($a$)
        MixColumn($a$)
        AddRoundKey($a$,$b$)
    $KR_r$ = $a$
```

Fig. 2. Pseudo code for the new AES key schedule

In Fig. 2, each Mk_j represents one byte of the cipher key. ByteSub, ShiftRow, MixColumn and AddRoundKey are the components of the AES round function. S[ ] is the S-box used in AES. Each KR_r represents a 128-bit round key.

3.3.2 Weakness in the New AES Key Schedule

We consider the following scenario. Consider that a 64-bit key being used in AES-192 and AES-256. And very likely the 64-bit key is concatenated to form the 192 and 256-bit key, respectively. Then the first 12 round keys for AES-192 and AES-256 would be identical. Now an attack could be applied to the last two rounds of AES-256.

4 Method to Resist the Related Cipher Attack

In Section 3, we applied the related-attack on SQUARE, AES and a new AES key schedule. AES is able to resist the attack. In this section, we introduce a general method to resist the related-cipher attack on block ciphers with flexible round number: relating the key schedule to the total round number. So when the same key is used to encrypt the same plaintext, the intermediate value after the *i*th round in the r round cipher should be quite different from that in the r' round cipher ($r \neq r'$).

The actual implementation that relates the key schedule to the total round number may vary from cipher to cipher. In the following example, we show how to relate the key schedule of SQUARE to the total round number. The original key schedule of SQUARE is maintained and additional modification is carried out on the subkeys.

After the original key schedule of SQUARE, we denote all the subkeys as $k_1,...,k_n$ (each one is one byte). The additional modification is carried out in this way:

```
for $i = 1$ to $n$ do
    $k_i = S_\gamma[k_i + S_\gamma[r]];$
```

where S_γ is the S-box used in SQUARE and r is the total round number. We expect that SQUARE with this modified key schedule could resist the related-cipher attack.

Since 8 round SQUARE is in use now, we suggest to keep the 8-round SQUARE the same as in [3], but those SQUARE with increased round number may adopt this strengthened key schedule.

5 Conclusion

In this paper, we introduced the related-cipher attack and applied this attack to some block ciphers with flexible round number. A Block cipher with flexible round number but with key schedule unrelated to the total round number is vulnerable to this attack. Care should be taken in designing ciphers with flexible features. A method to resist related cipher attack by relating the key schedule to the total round number is also suggested.

After introducing the related-cipher attack, we can classify the differential attack as related-message attack (the original differential cryptanalysis), related-key attack and related-cipher attack. Any combination of these attacks also gives a new attack. We believe that the cipher design should take all these attacks into consideration.

References

1. E. Biham and A. Shamir. Differential Cryptanalysis of the Data Encryption Standard, Springer-Verlag, 1993.
2. E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. In T. Helleseth, editor, Advances in Cryptology: Proc. of Eurocrypt '93, LNCS 765, pages 398-409, Springer-Verlag, 1994.
3. J. Daemen, L. Knudsen and V. Rijmen. The Block Cipher SQUARE. In E. Biham, editor, Fast Software Encryption – Proc. Forth International Workshop, Haifa, Israel, January 1997, LNCS 1267, pages 13-27, Springer Verlag, 1997.
4. C. Harpes and J.L. Massey. Partitioning Cryptanalysis. In E. Biham, editor, Fast Software Encryption: Forth International Workshop, Haifa, Israel, January 1997, LNCS 1267, pages 13-27, Springer Verlag, 1997.
5. L.R. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, Fast Software Encryption: Second International Workshop, Leuven, Belgium, 1994, LNCS 1008, pages 196-211, Springer Verlag, 1995.
6. L.R. Knudsen and M.J.B. Robshaw. Non-Linear Approximations in Linear Cryptanalysis. In U. Maurer, editor, Advancess in Cryptology – Eurocrypt'96, LNCS 1070, pages 224-235. Springer Verlag, 1995.
7. X. Lai. Higher order derivatives and differential cryptanalysis. In Proc. "Symposium on Communication, Coding and Cryptography", in honor of James L. Massey on the occasion of his 60'th birthday, Feb. 10-13, 1994, Monte-Verita, Ascona, Switzerland, 1994.

8. S.K. Langford and M.E. Hellman. Differential-linear cryptanalysis. In Y. G. Desmedt, editor, Advances in Cryptology – Proc. Crypto'94, LNCS 839, pages 17-26. Springer Verlag, 1994.
9. B.S. Kaliski Jr. and M.J.B. Robshaw. Linear Cryptanalysis Using Multiple Approximations. In Y. G. Desmedt, editor, Advances in Cryptology – Proc. Crypto'94, LNCS 839, pages 27-39. Springer Verlag, 1994.
10. J. Massey. SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm. In R. Anderson, editor, Fast Software Encryption – Proc. Cambridge Security Workshop, Cambridge, U.K., Dec. 9-11, 1993, LNCS 809, Springer-Verlag, pp.1-17. [See also: SAFER K-64: One Year Later. In B. Preneel, editor, Fast Software Encryption-- Proceedings of the Second International Workshop on Fast Software Encryption, Springer-Verlag, 1995, pp.212-241; and Strengthened Key Schedule for the Cipher SAFER, posted to the USENET newsgroup sci.crypt, September 9, 1995]
11. M. Matsui, Linear Cryptanalysis Method for DES Cipher. In T. Helleseth, editor, Advances in Cryptology – Proc. of Eurocrypt '93, LNCS 765, pages 386-397, Springer-Verlag, 1994.
12. L. May, M. Henricksen, W. Millan, G. Carter, and E. Dawson, Strengthening the Key Schedule of the AES. In Information Security and Privacy – Proc. of ACISP 2002, LNCS 2384, pages 226-240.
13. National Institute of Standards and Technology, Advanced Encryption Standard. Available at http:// csrc.nist .gov/encryption/aes/

A Chosen Plaintext Linear Attack on Block Cipher CIKS-1

Changhoon Lee[1], Deukjo Hong[1], Sungjae Lee[2], Sangjin Lee[1], Hyungjin Yang[1], and Jongin Lim[1]

[1] Center for Information Security Technologies(CIST)
Korea University, Anam Dong, Sungbuk Gu, Seoul, Korea
{crypto77,hongdj,sangjin,jilim}@cist.korea.ac.kr, {yangh}@korea.ac.kr
[2] Korea Information Security Agency(KISA)
Karag-dong, Songpa-gu, Seoul, Korea
{sjlee}@kisa.or.kr

Abstract. In this paper, we firstly evaluate the resistance of the reduced 5-round version of the block cipher CIKS-1 against linear cryptanalysis(LC). A feature of the CIKS-1 is the use of both Data-Dependent permutations(DDP) and internal key scheduing which consist in data-dependent transformation of the round subkeys. Taking into account the structure of CIKS-1 we investigate linear approximation. That is, we consider 16 linear approximations with $p = \frac{3}{4}$ for 16 parallel modulo 2^2 additions to construct one-round linear approximation and derive one-round linear approximation with the probability of $P = \frac{1}{2} + 2^{-17}$ by Piling-Up lemma. Also we estimate that the P is a valid probability of one-round approximation and achieve that the probability P for one-round approximation is better than $\frac{1}{2} + 2^{-17}$ through experiments. Then we construct 3-round linear approximation with $P = \frac{1}{2} + 2^{-17}$ using this one-round approximation and can attack the reduced 5-round CIKS-1 with 64-bit block by LC. In conclusion, we present that our attack requires about 2^{36} chosen plaintexts with a probability of success of 78.5 % and $\frac{1}{5} \times 2^{32} \times 2^{36} \approx 2^{65.7}$ encryption times to recover last round(5-round) key. In addition, we discuss a few improvements of the cipher CIKS-1.

Keywords: Block Cipher, Linear Cryptanalysis, Data-Dependent Permutation, CIKS-1(Cipher with Internal Key Scheduling)

1 Introduction

Data-Dependent operations appear to be an interesting cryptographic primitive. Data-Dependent Rotation(DDR) of them were used for the first time by Becker in IBM [5]. DDR gained recognition after they were used extensively by Rivest in RC5 [14]. And it has been shown [7] that mixed use of DDR with some other simple operations is an effective way to thwart linear cryptanalysis. Recently, RC6[15] and MARS[2] of AES candidates also composed DDR with integer multiplication. DDR can be interpreted as a particular case of controlled permutations(CP)[11]. The execution of the CP-operation $P_{n/m(V)}(X)$ consist in performing the fixed permutation $\prod_V$ on X where X is input data and V is controlling vector. We will explain CP operation obviously in the next section.

In this paper, we describe that the block cipher CIKS-1 introduced in Journal of cryptology (Vol.15 Num.1 winter 2002)[12] is based on Data-Dependent permutations(DDP) which are the basic cryptographic primitive to construct fast hardware-oriented cipher.(a feature of CIKS-1 is the use of both the Data-Dependent transformation of round subkeys and the key dependent DDP operations) And we analyze the security of the reduced 5-round version of the block cipher CIKS-1 against LC. Taking into account the structure of CIKS-1 we investigate linear approximation for it. That is, we consider 16 linear approximations with $p(=\frac{3}{4})$ for 16 parallel modulo 2^2 additions to construct one-round linear approximation and derive one-round linear approximation with probability value $P=\frac{1}{2}+2^{-17}$ by Piling-Up lemma [10]. Also we estimate that P is a valid probability for one-round approximation and achieve that the probability P for one-round approximation is better than $\frac{1}{2}+2^{-17}$ through experiments. Then we construct 3-round linear approximation with $P=\frac{1}{2}+2^{-17}$ using this one-round approximation and attack the reduced 5-round CIKS-1 with 64-bit block, 160-bit key by LC. If one persist in using CIKS-1 with original 256-bit key, then we also can attack full-round CIKS-1 with 256-bit key through the canonical extension of our attack. So we attack only the reduced 5-round CIKS-1 with 160-bit key by LC. In conclusion, we present that our attack requires about 2^{36} chosen plaintexts with a probability of success of 78.5% and $\frac{1}{5}\times 2^{32}\times 2^{36}\approx 2^{65.7}$ encryption times to recover last round(5-round) key. In addition, we discuss the improvements of CIKS-1.

This remainder of this paper is organized as follows. In Section 2 we summarize description of the cipher CIKS-1 and describe some approaches for achieving our attack and our attack on the reduced 5-round CIKS-1 against LC in Section 3 and conclude in Section 4.

2 Description of CIKS-1

In this section we describe a new block CIKS-1 based on CP-boxes performing data-dependent permutations.

2.1 Preliminary

We consider notations, definitions and properties of CP-boxes. And most of notations and definitions used in this paper are them used in [12].

Definition 1. *Let two operands input vector $X=(x_0,...,x_{n-1})$ and controlling vector $V=(v_0,...,v_{m-1})$ be given and let be output vector $Y=(y_0,...,y_{n-1})$ and let be order set $\prod_V=\{\Pi_0,\Pi_1,...,\Pi_{2^m-1}\}$ is a family of fixed permutations of some set of n bits (or CP -modifications). Then, the execution of CP operation $P_{n/m(V)}(X)$ consist in performing the fixed permutation $\prod_V$ on X, i.e, $Y=P_{n/m(V)}(X)=\prod_V(X)$*

A feature of CP-box $P_{n/m}$ is shown in Fig.1.

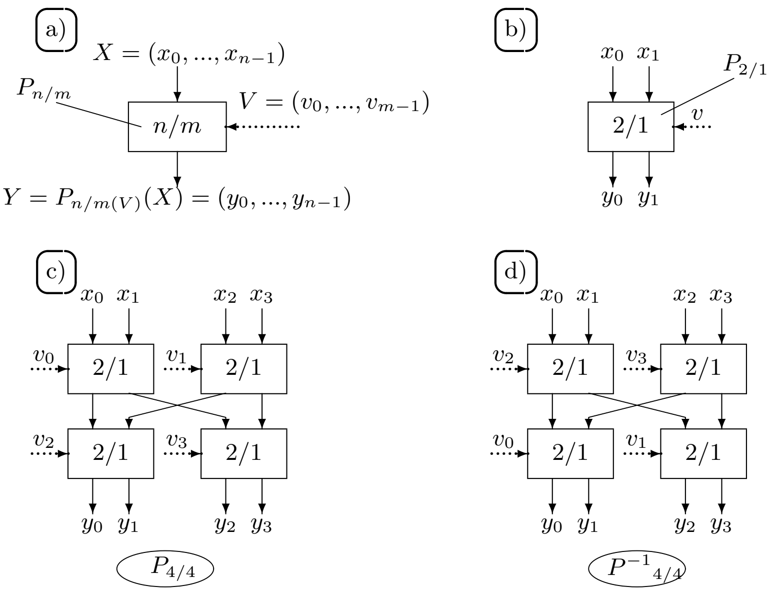


Fig. 1. Structures of the basic CP-box permutations

In the general case the value V is assumed to be dependent on encrypted data and(or) Key. CP-boxes can be constructed as a superposition of the standard elementary $P_{2/1}$-boxes shown in Fig.1. A $P_{2/1}$-box is controlled by one bit v. If $v = 0$, it swaps two input bits otherwise(if $v = 1$), the bits are not swapped.

Definition 2.
1. Let a $P_{n/m}$-box be given. Suppose for arbitrary $h \leq n$ input bits $x_1, x_2, ..., x_h$ and arbitrary h output bits $y_1, y_2, ..., y_h$ there is at least one CP-modification moving x_i to y_i for all $i = 1, 2, ..., h$. Such a $P_{n/m}$-box is called a CP-box of order h.
2. A strict CP-box is a $P_{n/m}$-box in which all CP-modification are pairwise unequal.
3. CP-boxes $P_{n/m}$ and $P^{-1}{}_{n/m}$ are mutual inverses, if for all possible values of the vector V the corresponding CP-modifications $\prod_V$ and $\prod_V^{-1}$ are mutual inverses.

The cipher CIKS-1 is composed of variants of the CP-boxes in Fig.2, 3 and we explain a detailed figure of the cipher CIKS-1 in next subsection.

2.2 The Block Cipher CIKS-1

The design rationale for block cipher CIKS-1 is based to extensive use of the CP-box permutations which are fast and inexpensive in hardware implementation. CIKS-1 is a 64-bit block iterated cipher with r(=8) rounds and a 256-bit Master key K. The master key K is divided into 8 32-bit subkeys to be injected into each round transformation with internal key scheduling [12].

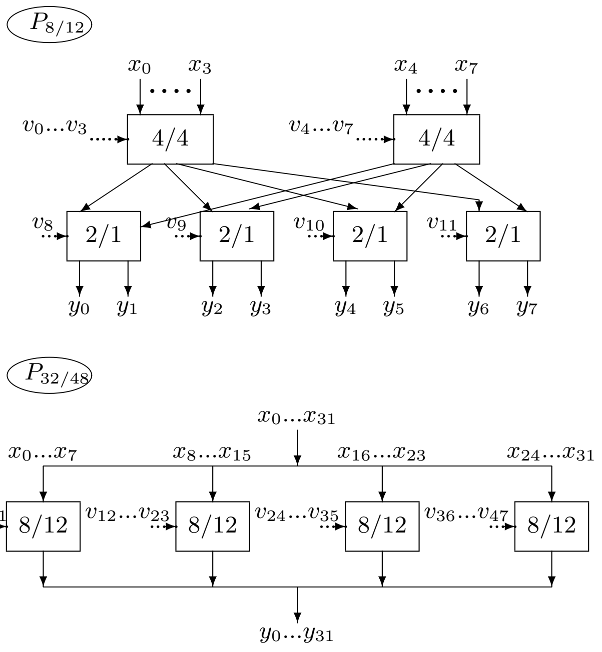
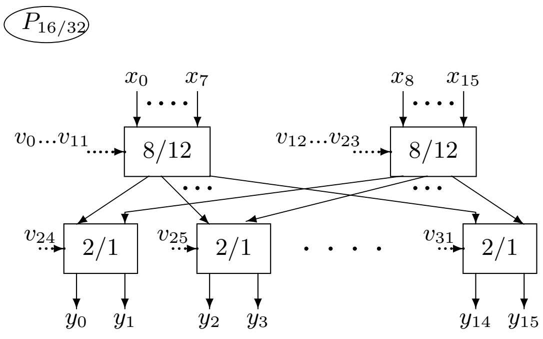


Fig. 2. Structures of the basic CP-box permutations

One Round Structure.
Here, we describe the description of one round structure of CIKS-1. The cipher CIKS-1 is represented in Fig 4. In addition to CP-box permutations, one encryption (decryption) round of CIKS-1 uses fixed permutations ($\prod_1$, $\prod_2$, rotations by 7bits), one XOR operation, and 16 parallel modulo 2^2 additions (subtrations) denoted as a single operation "$+ ... +$"("$- ... -$").

- Operations

1 Operation XOR combines the right data subblock with current round subkey after the latter is permuted in dependence on the left subblock.
2 Sixteen parallel modulo 2^2 additions (subtractions) are used to mix together values of the left and right subblocks instead of one modulo 2^{32} addition (subtraction). To execute operation "$+...+$"$(or$"$-...-$") each of two 32-bit operands is divided into 16 2-bit operands and then 16 modulo 2^2 additions (subtractions) are performed simultaneously on respective pairs of the 2-bit operands.
3 Fixed permutations improve data diffusion.
(1) Permutation $\prod_1$ is described by the following formula:
$V' = (v'_0, ..., v'_{79}) = \prod_1(L|L'|S') = \prod_1(l_0, ..., l_{31}, l_{16}, ..., l_{31}, s'_0, ..., s'_{31})$
$= (l_8, ..., l_{31}, s'_0, ..., s'_7, l_{16}, ..., l_{31}, l_0, ..., l_7, s'_8, ..., s'_{31})$,
where $S' = (s'_0, ..., s'_{31})$ is the value of the left subblock permuted in dependence on the current round subkey.



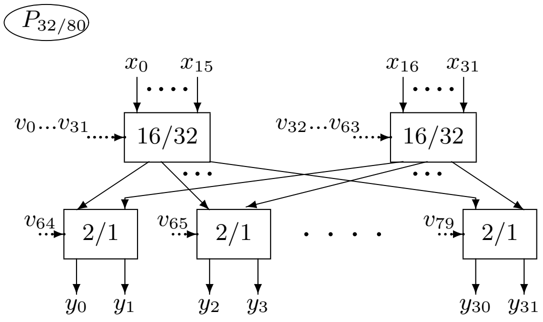


Fig. 3. Structures of the basic CP-box permutations

(2) Permutation $\prod_2$ is described by the following formula:

$$\begin{aligned} V'' &= (v''_0, ..., v''_{79}) = \prod\nolimits_2(S''|L^*|L'') = \prod\nolimits_2(s''_0, ..., s''_{31}, l^*_0, ..., l^*_{31}, l^*_0, ..., l^*_{15}) \\ &= (s''_{16}, ..., s''_{23}, l^*_0, ..., l^*_3, s''_{24}, ..., s''_{31}, l^*_4, ..., l^*_{15}, s''_0, ..., s''_7, l^*_{16}, ..., l^*_{19}, \\ &\qquad s''_8, ..., s''_{15}, l^*_{20}, ..., l^*_{31}, l^*_0, ..., l^*_{15}), \end{aligned}$$

where $S'' = (s''_0, ..., s''_{31}) = P_{32/48(V_K >>> 7)}(L^*)$.

In a word, CIKS-1 is an eight-round iterated cipher with a 256-bit secret key K represented as a set of 32-bit subkeys $K^1, \cdots, K^8$ (*i.e*, $K = K^1 \| \cdots \| K^8$). CIKS-1 can be described by the following algorithm.

- Algorithm

Step 1 Input: 64-bit plaintext $P_L \| P_R$ represented as concatenation of two 32-bit subblocks P_L and P_R.

Step 2 For i=1 to 7 do $\{P_L \| P_R$:=Crypt$(P_L \| P_R)$;$P_L \leftrightarrow P_R\}$, where Crypt means a round encryption procedure and "$\leftrightarrow$" denotes swapping blocks.

Step 3 $P_L \| P_R$:=Crypt$(P_L \| P_R)$

Step 4 Output: ciphertext $P_L \| P_R$

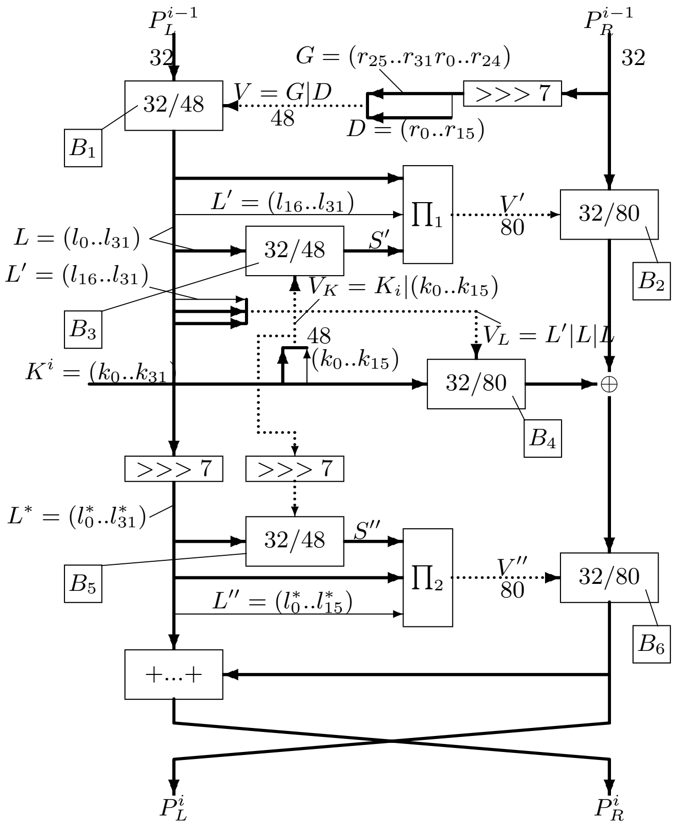


Fig. 4. The *i*th-round transformation with internal key scheduling in the CIKS-1

3 Linear Cryptanalysis on CIKS-1

The most powerful attack with Differential Cryptanalysis(DC)[3] on block ciphers is LC[10] which was introduced by Matsui. The attack explores linear relations between plaintext, ciphertext and subkey bits. Linear approximations for an iterated cipher are usually made by combining approximations for each round. We find the following linear approximation which holds with the probability $p \neq 1/2$ for randomly given plaintext P and the corresponding ciphertext C and analyze the CIKS-1 cipher using the Algorithm 2 in [10].

$$P[i_1, i_2, ..., i_a] \oplus C[j_1, j_2, ..., j_b] = K[k_1, k_2, ..., k_c] \tag{1}$$

where $i_1, i_2, ..., i_a$, $j_1, j_2, ..., j_b$ and $k_1, k_2, ..., k_c$ denote fixed bit locations and $P[i_1, i_2, ..., i_a] = P[i_1]\oplus, ..., \oplus P[i_a]$.

3.1 The Approach

We explain main approaches to evaluate resistance of CIKS-1 against LC.

- Our Approaches

1. We easily know the following property of CP-boxes(permutations). if the number of "1"(or "0") in input data of CP-boxes is n, then the number of "1"(or "0") in output data of CP-boxes is also equal to n without regard to control vector V. Thus we know that $I[i_0, ..., i_{n-1}] = O[o_0, ..., o_{n-1}]$ with probability $p = 1$ where I(or O) denotes the input(or output) data.

2. We shall consider 16 linear approximations for 16 parallel modulo 2^2 additions to construct one-round linear approximation of CIKS-1. Let $L = (l_0, ..., l_{31})$ and $R = (r_0, ..., r_{31})$ be 32-bit operands of operation " $+ ... +$ " and let $G(L, R)(m_0, .., m_{31})$ be output of " $+ ... +$ " for two input L and R. Then,

$$\begin{aligned} L[l_0, l_1] \oplus R[r_0, r_1] &= G(L, R)[m_0, m_1] \\ L[l_2, l_3] \oplus R[r_2, r_3] &= G(L, R)[m_2, m_3] \\ &\cdots\cdots\cdots\cdots \\ L[l_{30}, l_{31}] \oplus R[r_{30}, r_{31}] &= G(L, R)[m_{30}, m_{31}] \end{aligned}$$

   We easily can achieve that each probability value of 16 linear approximations above is equal to $p = \frac{3}{4}$ and then using Piling-Up lemma [10] we can derive the following one-round linear approximation of CIKS-1 with probability$P = \frac{1}{2} + 2^{-17}$.

$$L[l_0, ..., l_{31}] \ \oplus \ R[r_0, ..., r_{31}] = G(L, R)[m_0, ..., m_{31}] \qquad (2)$$

   Also we estimated that the P is a valid probability for the equation(2) and could achieve that the probability P for one-round approximation is better than $1/2 + 2^{-17}$ through experiments.

3. We easily know that if we choose the following plaintexts(•) , then the linear probability for the equation(2) is equal to $P = 1$ in the first round.

   • (P_L^1, P_R^1)=($(l_0, ..., l_{31})$, 32-bit random vector) where $l_{2i+1} = 0$ if for all i, $0 \leq i \leq 15$.
   Since each probability of 16 linear approximations for 16 parallel modulo 2^2 additions is equal to $p = 1$ in the case(•).

3.2 The Chosen Plaintext Linear Attack on the Reduced 5-Round CIKS-1

In this subsection, we introduce a 3-round linear approximation used in our attack on CIKS-1 and we attack the reduced 5-round CIKS-1 with 160-bit key by using it. But if one persists in using CIKS-1 with original 256-bit key, then we also can attack full-round CIKS-1 with 256-bit key through the canonical extension of our attack. So, we analysis only 5-round CIKS-1 with 160-bit key by LC and our attack is described in Fig.5, 6, 7. Let $P_L^i(P_R^i)$ denote the 32-bit left(right) input in the i-th round in CIKS-1. Then we choose plaintexts(●) presented in previous section 4.1 and then using approach 1,2 of the previous section we derive the 3-round linear approximation with maximal probability for 5-round CIKS-1. This linear approximation is shown in Fig.5, 6, 7. We denote $A[a_0, ..., a_{31}]$ by $A[all]$ for convenience where $A = (a_0, ..., a_{31})$ means 32-bit random vector and define linear characteristic probability LCP_i for ith round. [1] ($i.e$, $LCP_i = |2P_i - 1|^2$ where P_i is the probability for linear approximation of ith round.)

To begin with, let $L^i(R^i)$be the left(right) input of " + ... + " in the ith round and let B_j^i is a input(or output) data of $B_j box$ in the ith round. we can observe that $P_L^1[all] = B_1^1[all] = L^1[all]$ and that $P_R^1[all] = B_2^1[all] \Longrightarrow B_6^1[all] = K^1[all] \oplus P_R^1[all] = R^1[all]$ in the first round.
Then we can know $L^1[all] \oplus R^1[all] = G^1(L,R)[all]$ with $P_1 = 1$. Therefore we can construct the linear approximation of the first round. (See Fig.5)

$$\begin{aligned} \text{1st round : } & P_R^1[all] \oplus P_L^2[all] = K^1[all] \text{ with } P_{1'} = 1 \\ & P_L^1[all] \oplus P_R^1[all] \oplus P_R^2[all] = K^1[all] \text{ with } P_1 = 1 \end{aligned}$$

Also, we can observe that $P_L^2[all] = K^1[all] \oplus P_R^1[all] = B_1^2[all] = L^2[all]$ and that $P_R^2[all] = P_L^1[all] \oplus K^1[all] \oplus P_R^1[all] = B_2^2[all] \Longrightarrow B_6^2[all] = K^2[all] \oplus B_2^2[all] = R^2[all]$ in the 2nd round. Then $L^2[all] \oplus R^2[all] = G^2(L,R)[all]$ with $P_2 = 1/2 + 2^{-17} \Longleftrightarrow (K^1[all] \oplus P^1{}_R[all]) \oplus (K^1[all] \oplus K^2[all] \oplus P^1{}_R[all] \oplus P_L^1[all]) = P_L^1[all] \oplus K^2[all] = G^2(L,R)[all]$. Therefore we can construct the linear approximation of the second round. (See Fig.5)

$$\begin{aligned} \text{2nd round : } & P_R^2[all] \oplus P_L^3[all] = K^2[all] \text{ with } P_{2'} = 1 \\ & P_L^2[all] \oplus P_L^3[all] \oplus P_R^3[all] = 0 \text{ with } P_2 = 1/2 + 2^{-17} \\ & \text{and } LCP_2 = 4 \times 2^{(-17)^2} = 2^{-32} \end{aligned}$$

and we can observe that $P_R^3[all] = P_L^1[all] \oplus K^2[all] = B_2^3[all] \Longrightarrow B_6^3[all] = K^3[all] \oplus B_2^3[all] = K^3[all] \oplus K^2[all] \oplus P_L^1[all] = R^3[all]$ in the 3rd round. Then we can construct the linear approximation of the third round. (See Fig.6)

$$\text{3rd round : } P_R^3[all] \oplus P_L^4[all] = K^3[all] \text{ with } P_{3'} = 1$$

Hence these one-round approximations lead to the following 3-round approximation for 5-round CIKS-1. (See Fig.5, 6)
That is, we calulate that $(P_R^1[all] \oplus P_L^2[all] = K^1[all]) \oplus (P_L^1[all] \oplus P_R^1[all] \oplus P_R^2[all] = K^1[all]) \oplus (P_R^2[all] \oplus P_L^3[all] = K^2[all]) \oplus (P_L^2[all] \oplus P_L^3[all] \oplus P_R^3[all] = 0) \oplus (P_R^3[all] \oplus P_L^4[all] = K^3[all])$ and then we get the following relation.

$$\begin{gathered} P_L{}^4[all] = K^2[all] \oplus K^3[all] \oplus P_L^1[all] \text{ with a probability } P = 1/2 + 2^{-17} \\ \text{by Piling-Up lemma and } LCP = LCP_2 = 2^{-32} \end{gathered}$$

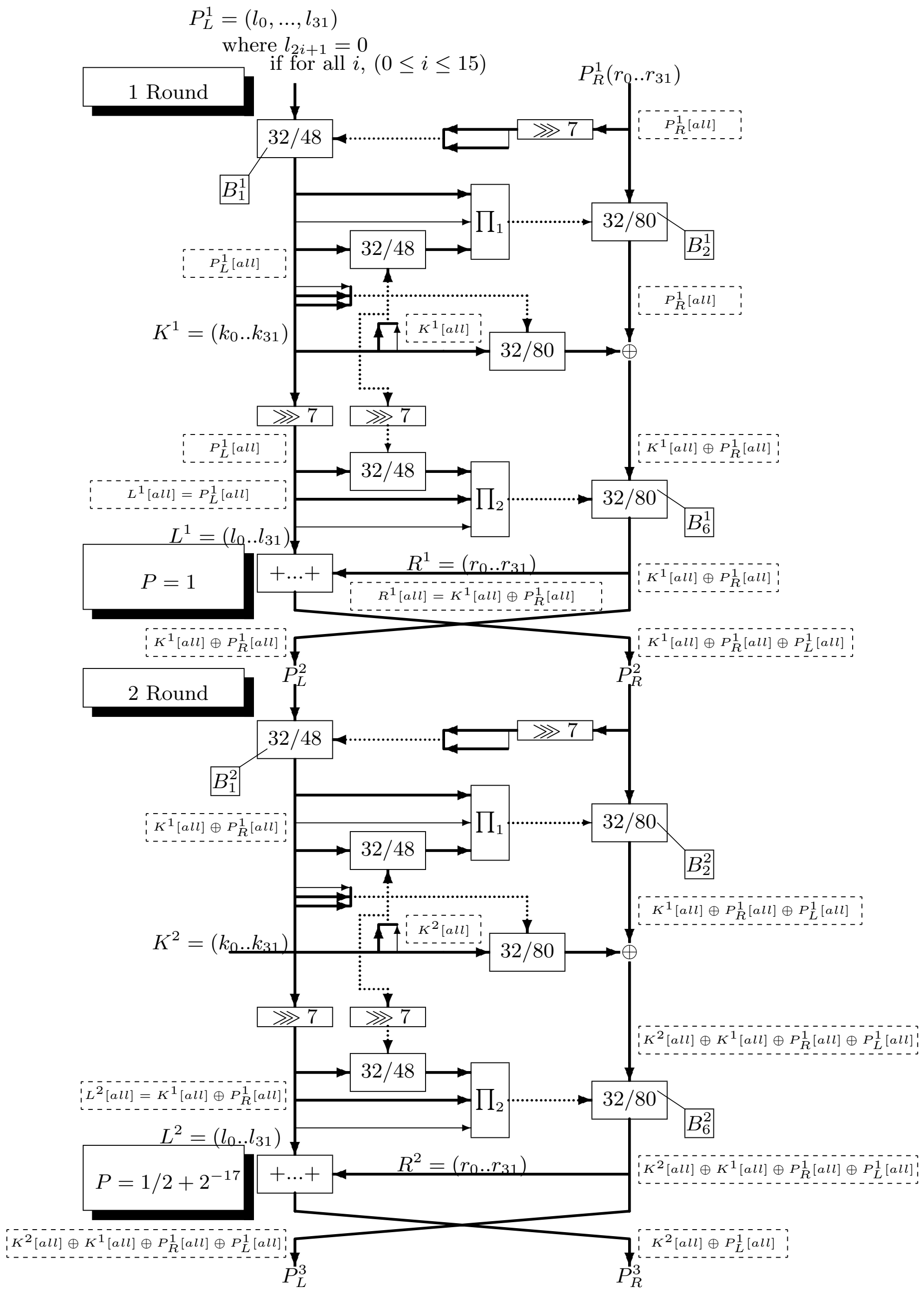

$P_L^1 = (l_0, ..., l_{31})$
where $l_{2i+1} = 0$
if for all i, $(0 \leq i \leq 15)$
$P_R^1(r_0..r_{31})$
1 Round
32/48
$\ggg 7$
$P_R^1[all]$
B_1^1
$\prod_1$
32/80
B_2^1
$P_L^1[all]$
$P_R^1[all]$
$K^1 = (k_0..k_{31})$
$K^1[all]$
$\ggg 7$
$P_L^1[all]$
$K^1[all] \oplus P_R^1[all]$
$L^1[all] = P_L^1[all]$
$\prod_2$
B_6^1
$L^1 = (l_0..l_{31})$
$P = 1$
+...+
$R^1 = (r_0..r_{31})$
$R^1[all] = K^1[all] \oplus P_R^1[all]$
$K^1[all] \oplus P_L^1[all] \oplus P_R^1[all]$
$K^1[all] \oplus P_R^1[all]$
$K^1[all] \oplus P_R^1[all] \oplus P_L^1[all]$
P_L^2
P_R^2
2 Round
B_1^2
B_2^2
$K^1[all] \oplus P_R^1[all]$
$K^1[all] \oplus P_R^1[all] \oplus P_L^1[all]$
$K^2 = (k_0..k_{31})$
$K^2[all]$
$K^2[all] \oplus K^1[all] \oplus P_R^1[all] \oplus P_L^1[all]$
$L^2[all] = K^1[all] \oplus P_R^1[all]$
B_6^2
$L^2 = (l_0..l_{31})$
$P = 1/2 + 2^{-17}$
$R^2 = (r_0..r_{31})$
$K^2[all] \oplus K^1[all] \oplus P_R^1[all] \oplus P_L^1[all]$
$K^2[all] \oplus K^1[all] \oplus P_R^1[all] \oplus P_L^1[all]$
$K^2[all] \oplus P_L^1[all]$
P_L^3
P_R^3

Fig. 5. Our Attack on CIKS-1

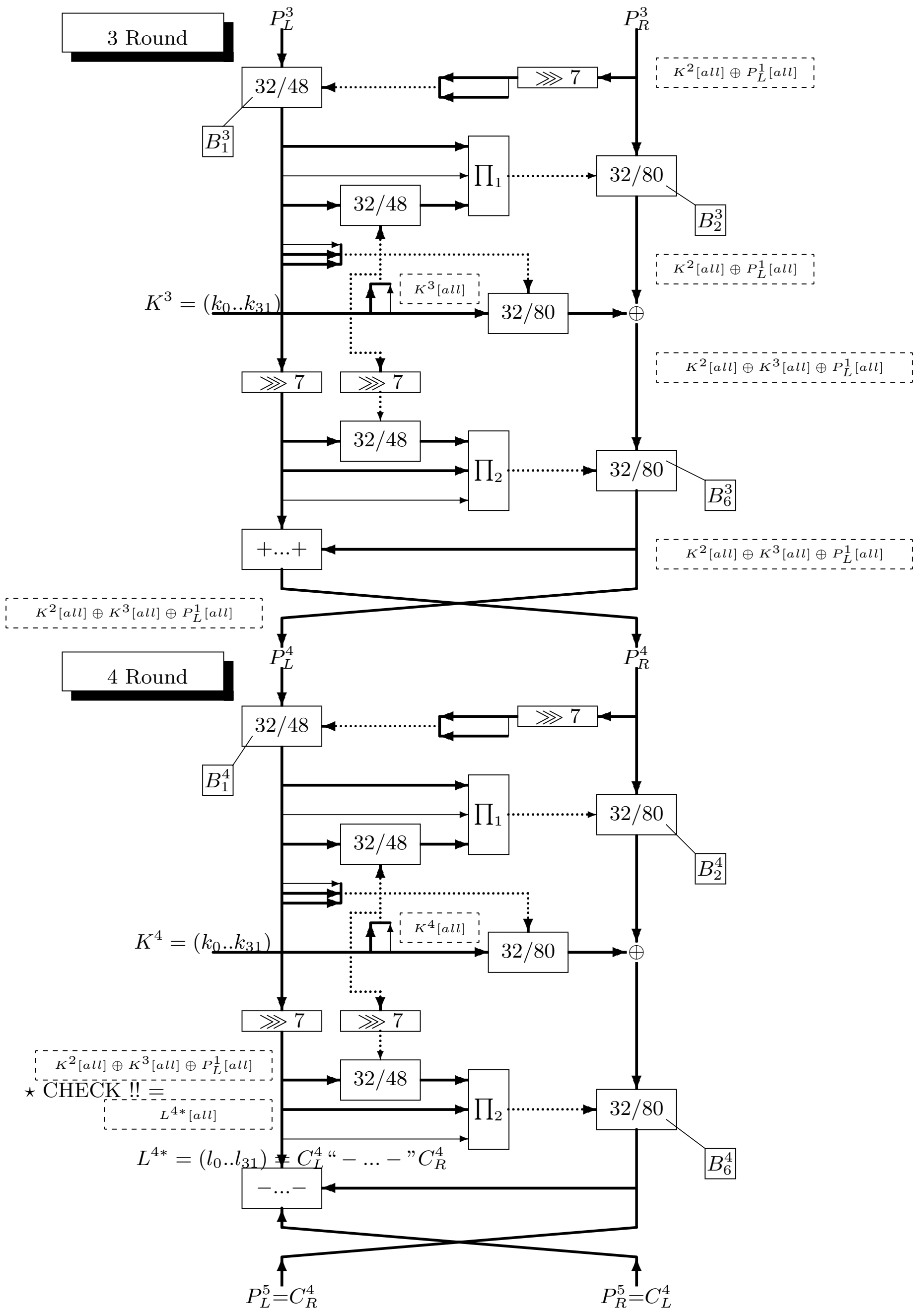


Fig. 6. Our Attack on CIKS-1

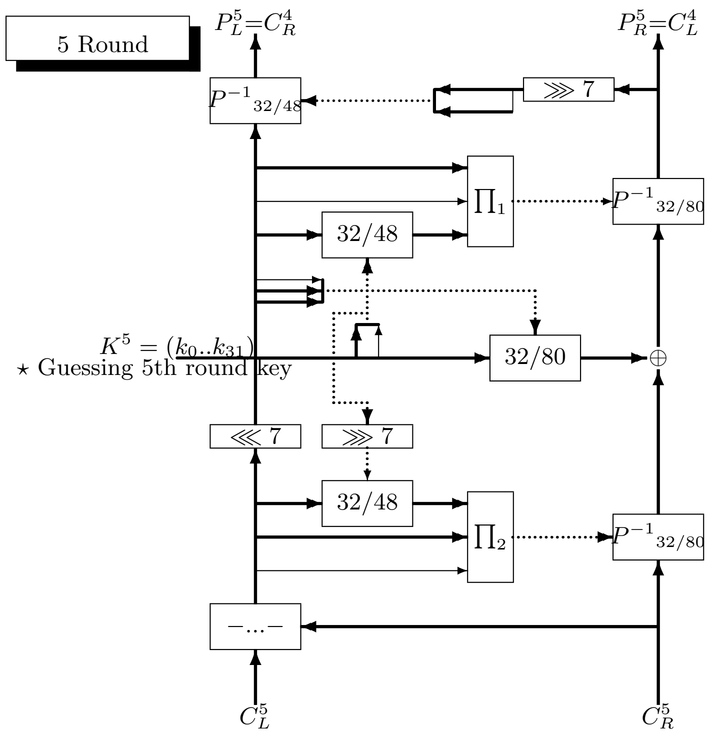


Fig. 7. Our Attack on CIKS-1

Then we apply this approximation to our attack on CIKS-1. Our attack method is followed as: (See Fig.5, 6, 7)

- Attack Algorithm

1. Acquire 2^{36} Chosen plaintext/ciphertext-pairs such that
   - $(P_L^1, P_R^1) = ((l_0, ..., l_{31})$, (32-bit random vectors $/= 0$)) where $l_{2i+1} = 0$ if for all i, $0 \le i \le 15$.
2. For each of 2^{32} guessed 5th-round key do:
   For each of 2^{36} plaintext/ciphertext-pair do:
   After decrypting 5th-round,
   (1) Compute 4 round output C_L^4 and C_R^4 and L^{4*}
   (2) Compute $L^{4*}[all]$ and if $(L^{4*}[all] = 0)$, $T = T^{++}$
   where T: the number of "$L^{4*}[all] = 0$"
   (3) Let $T_{max}(T_{min})$ be the maximal(minimal) value of all T's.
   - If $|T_{max} - N/2| > |T_{min} - N/2|$ then adopt the candidate corresponding to T_{max} and guess $L^{4*}[all] = P_L^4[all] = P_L^1[all] \oplus K^2[all] \oplus K^3[all] = 0$ (when $P > 1/2$) or 1 (when $P < 1/2$)
   - If $|T_{max} - N/2| < |T_{min} - N/2|$ then adopt the candidate corresponding to T_{min} and guess $L^{4*}[all] = P_L^4[all] = P_L^1[all] \oplus K^2[all] \oplus K^3[all] = 1$ (when $P > 1/2$) or 0 (when $P < 1/2$)

Thus if a linear attack can be mounted on CIKS-1 with 5 rounds using 2^{36} chosen plaintexts, then our attack will succeed in finding last round subkey with a probability of 78.5% since $4 \cdot |P - (\frac{1}{2})|^{-2} = 2^{36}$ data complexity.[10] Hence we present that our attack requires about 2^{36} chosen plaintexts to recover 5th-round key and estimate that time complexity is about $\frac{1}{5} \times 2^{32} \times 2^{36} \approx 2^{65.7}$.

4 Conclusion

In this paper we have evaluated the resistance of the reduced 5-round CIKS-1 against LC. and have showed that our attack requires 2^{36} chosen plaintexts to recover 5th-round key with a probability of success of 78.5%. Also, Using the properties of CP-boxes and Matsui's ideas, we have presented efficient linear approximation on CIKS-1 and introduced "Attack Algorithm ". In result, we have achieved that data complexity 2^{36} and time complexity is about $2^{65.7}$. As stated in previous section 3.2, the block cipher CIKS-1 has weak points in view of our attack and our approaches can be applied to other block cipher with DDP. Therefore we have to keep a our approaches in mind to design a secure block cipher with DDP. If one can solve these weakness for our attack then the block cipher CIKS-1 is worth exploiting in view of efficiency of cipher and we will study the cipher CIKS-1 in more detail and report new results in the near future.

References

1. K. Aoki and K. Ohta, *Strict Evaluation of the Maximum Average of Differential Probability and the Maximem Average of Linear Probability*, IEICE Transcations fundamentals of Elections, Communications and Computer Sciences, No.1, 1997, pp 2-8.
2. C.Burwick, D.Coppersmith, E.D'Avingnon, R.Gennaro et al. *MARS*, Proceedings of the 1st Advanced Encryption Standard Candidate Conference, Venture,California,Aug.20-22,1998.
3. E.Biham and A.Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, New York, 1993.
4. E.Biham, *On Matsui's Linear Cryptanalysis*, Advanced in cryptology , Eurocrypt'94, Springer-Verlag, 1994.
5. W.Becker, *Method and system for machine enciphering and deciphering*, U.S. patent number 4157454, 1979.
6. J.Borst, B.Preneel, J.Vandewalle, *Linear Cryptanalysis of RC5 and RC6*, FSE'99, Springer-Verlag, 1994.
7. B.Kaliski, Y.L.Yin, *On differential linear Cyptanalysis of RC5 encryption algorithm*,Advanced in cryptology, CRYPTO'95, 1995.
8. L.R.Knudsen and J.E. Mathiassen, *A Chosen-Plaintext Linear Attack on DES*, FSE'2000, 2000.
9. M.Matsui, *A new method for known plaintext attack of FEAL cipher*, Advanced in cryptology , Eurocrypt'92, Springer-Verlag, 1992.
10. M.Matsui, *Linear cryptanalysis method for DES cipher*, Advanced in cryptology , Eurocrypt'93, Springer-Verlag, 1993.

11. A.A.Moldovyan, N.A.Moldovyan, *A method of the cryptographical transformation of binary data blocks*, Russian patent number 2141729 Bull.no.32,1999.
12. A.A.Moldovyan, N.A.Moldovyan, *A cipher based on data-dependent permutations*, Journal of Cryptology, Vol.15, Num. 1, Winter 2002.
13. J.Nakahara Jr, B.Preneel, J.Vandewalle,*Linear cryptanalysis of reduced-round versions of the SAFER*, Fast Software Encryption'96, Springer-Verlag, 1996.
14. R.L.Rivest, *The RC5 encryption algorithm*, Fast Software Encryption'94, Springer-Verlag, 1995.
15. R.L.Rivest,M.Robshow,R.Sidney, and Y.L. Yin *The RC6 Block cipher*, Proceedings of the 1st Advanced Encryption Standard Candidate Conference, Venture, California, Aug.20-22,1998.

Ideal Threshold Schemes from Orthogonal Arrays

Josef Pieprzyk and Xiam-Mo Zhang

Centre for Advanced Computing – Algorithms and Cryptography
Department of Computing, Macquarie University
Sydney, NSW 2109, Australia
{josef,xianmo}@ics.mq.edu.au

Abstract. The work investigates the design of ideal threshold secret sharing in the context of cheating prevention. We showed that each orthogonal array is exactly a defining matrix of an ideal threshold scheme. To prevent cheating, defining matrices should be nonlinear so both the cheaters and honest participants have the same chance of guessing of the valid secret. The last part of the work shows how to construct nonlinear secret sharing based on orthogonal arrays.

Keywords: Threshold Secret Sharing, Orthogonal Arrays, Cheating Detection, Nonlinear Functions.

1 Introduction

Secret sharing is one of basic cryptographic algorithms that is of great importance for cryptographic services where actions are controlled by groups. It is a well known fact that linear secret sharing is vulnerable to cheating attacks. In these attacks, dishonest participants submit forged shares to the combiner who returns an invalid secret. The cheaters, knowing the invalid secret and their valid shares, are able to recover the valid secret. As the result, the cheaters hold the valid secret while the honest participants are left with the invalid one distributed by the combiner.

This paper explores how (nonlinear) orthogonal arrays can be used to build ideal threshold schemes that are immune against cheating. The immunity against cheating springs from the fact that, in nonlinear secret sharing, a cheater is unable to obtain the valid secret. More precisely, the probability of guessing the valid secret by the cheater and honest participants are the same.

This work is structured as follows. The basic concepts of secret sharing are introduced in Section 2. In Section 3, we define a perfect secret sharing scheme using defining matrices. An ideal secret sharing scheme is a perfect scheme for which the set of secrets and the set of shares have the same size. We investigate some properties of ideal secret sharing in Section 4. In Section 5, we introduce orthogonal arrays and show how they can be used to construct ideal threshold schemes. In Section 6, we study properties of such ideal threshold schemes and their applications for cheating prevention. In Section 7 we demonstrate the existence of orthogonal arrays that are defining matrices of ideal threshold schemes.

Section 8 examines how cheating attacks can be prevented in secret sharing based on orthogonal arrays and constructions of secret sharing immune against cheating are given. Conclusions close the work.

2 Access Structures

Secret sharing is a method to share a secret among a set of participants $\mathbf{P} = \{P_1, \ldots, P_n\}$. Let $\mathbf{K}$ denote the set of *secrets* and $\mathbf{S}$ denote the set of *shares*. The secret sharing includes two algorithms: the distribution algorithm (dealer) and the recovery algorithm (combiner). The dealer assigns shares $s_1, \ldots, s_n$ to all the participants $P_1, \ldots, P_n$, respectively, or in other words, it creates the secret sharing system. After that the participants collectively hold the secret until there is a big enough subset of participants who wish to recover the secret by calling the recovery algorithm. Assume that the currently active participants are $P_{j_1}, \ldots, P_{j_\ell}$ and that they submit their shares to the combiner in order to recover the secret. Their shares $s_{j_1}, \ldots, s_{j_\ell}$ can determine a secret if and only if $\{P_{j_1}, \ldots, P_{j_\ell}\}$ is a qualified subset of $\mathbf{P}$, i.e., the set of currently active participants belongs to the access structure Γ. It turns out that any access control is *monotone* or any superset of qualified set of participants belongs to the access structure, or more precisely

$$\text{if } \mathcal{A} \in \Gamma \text{ and } \mathcal{A} \subseteq \mathcal{B} \subseteq \mathbf{P} \text{ then } \mathcal{B} \in \Gamma \tag{1}$$

We can describe secret sharing with the access structure Γ by an $m \times (n+1)$ matrix M^*, as shown in [2,3]. The matrix M^* has $n+1$ columns indexed by $0, 1. \ldots, n$. The number m of rows of M^* depends on a particular scheme. We index the m rows by $1, \ldots, m$. For a row of M^*, the entry in the 0th position holds a secret and the entry in the ith position ($i = 1, \ldots, n$) contains the corresponding share of P_i. Denote the entry on the ith row and the jth column of M^* by $M^*(i,j)$. The matrix M^* is called a *defining matrix* of secret sharing with the access scheme Γ. The matrix M obtained from M^* by removing the 0th column is called the *associated matrix* of the scheme.

The dealer works in two stages. In the first stage, it creates the defining matrix M^* for secret sharing with the access structure Γ. The matrix is made public. In the second stage, the dealer randomly chooses a row of the matrix M^*. Let the row chosen be indexed by the integer i. The secret is $K = M^*(i,0)$ and shares are $s_j = M^*(i,j)$, $j = 1, \ldots, n$. The shares are distributed to the corresponding participants via secure channels.

An access structure $\Gamma = \{\mathcal{A} \mid \#\mathcal{A} \geq t\}$ is called a (t,n)-*threshold access structure*, where $\#X$ denotes the cardinality of the set X (i.e., the number of elements in the set X) and the integer t is called the *threshold* of secret sharing, where $t \leq n$. Secret sharing schemes with the (t,n)-threshold access structure are called (t,n)-threshold schemes.

It should be noticed that a defining matrix uniquely determines a secret sharing scheme but a secret sharing scheme has more defining matrices. Permuting the rows of a defining matrix of secret sharing does not give a new scheme.

Clearly, two secret sharing schemes are considered to be the same if the defining matrix of the one can be obtained from the other by permuting the rows of its defining matrix. Permuting the columns of defining matrices of secret sharing is equivalent to changing the indices of participants. In other words, access structures of secret sharing with permuted columns are different but one access structure can be derived from the other by permuting the participants. For this reason, we do not regard the the resulting scheme as a new one. We say that the resulting scheme is *equivalent* to the original one.

It should be pointed out once again that a defining matrix of a secret sharing scheme is public. The dealer chooses at random a single row of the matrix. The shares are communicated to the corresponding participants via secure channels so the share s_i is known to the participant P_i only ($i = 1, \ldots, n$).

3 Perfect Secret Sharing

We say that secret sharing with the access structure Γ is perfect if the following two conditions are satisfied:

(1) If $\mathcal{A} \in \Gamma$ then the participants in $\mathcal{A}$ can uniquely determine the secret by pooling their shares together.
(2) if $\mathcal{A} \notin \Gamma$ then the participants from $\mathcal{A}$ can determine nothing about the secret (in an information theoretic sense).

As argued in [2], Conditions (1) and (2) can be translated into conditions that need to be satisfied in the context of the defining matrix.

(a) Let $\mathcal{A} \in \Gamma$. If $M^*(i,j) = M^*(i',j)$ for every $P_j \in \mathcal{A}$ then $M^*(i,0) = M^*(i',0)$.
(b) Let $\mathcal{A} \notin \Gamma$. For any $1 \le i_0 \le m$ and any $K \in \mathbf{K}$, there exists some i with $1 \le i \le m$ such that $M^*(i,j) = M^*(i_0,j)$ for all $P_j \in \mathcal{A}$ and $M^*(i,0) = K$.
(b') Let $\mathcal{A} = \{P_{j_1}, \ldots, P_{j_\ell}\} \notin \Gamma$. For any $s_{j_1}, \ldots, s_{j_\ell} \in \mathbf{S}$ and any $K \in \mathbf{K}$,

$$\#\{i \mid M^*(i,j_u) = s_{j_u} \text{ for all } P_{j_u} \in \mathcal{A} \text{ and } M^*(i,0) = K\}$$

is independent to the choice of K.

It is easy to verify that (b') implies (b). For the case of a (t,n)-threshold scheme, Conditions (a), (b), and (b') can be rewritten as follows:

(c) Let $\#\mathcal{A} \ge t$. If $M^*(i,j) = M^*(i',j)$ for every $P_j \in \mathcal{A}$ then $M^*(i,0) = M^*(i',0)$.
(d) Let $\#\mathcal{A} < t$. For any $1 \le i_0 \le m$ and any $K \in \mathbf{K}$, there exists some i with $1 \le i \le m$ such that $M^*(i,j) = M^*(i_0,j)$ for all $P_j \in \mathcal{A}$ and $M^*(i,0) = K$.
(d') Let $\mathcal{A} = \{P_{j_1}, \ldots, P_{j_\ell}\}$ with $\ell < t$. For any $s_{j_1}, \ldots, s_{j_\ell} \in \mathbf{S}$ and any $K \in \mathbf{K}$,

$$\#\{i \mid M^*(i,j_u) = s_{j_u} \text{ for all } P_{j_u} \in \mathcal{A} \text{ and } M^*(i,0) = K\}$$

is independent to the choice of K.

Similarly, (d') implies (d).

Definition 1. *A secret sharing scheme satisfying (a) and (b) is called* weakly perfect*, while it is called* perfect *if it satisfies (a) and (b') [2]. Alternatively, a (t,n)-threshold scheme satisfying (c) and (d) is called* weakly perfect*, while it is called* perfect *if it satisfies (c) and (d').*

Threshold schemes were first introduced by Blakley [1] and Shamir [6]. Ito et al [5] generalized threshold secret sharing for arbitrary monotonic access structures.

4 Ideal Secret Sharing

Given an access structure Γ. A set $\mathcal{A} \in \Gamma$ is called *minimal* if all proper subsets of $\mathcal{A}$ do not belong to Γ. It is easy to see that $\mathcal{A}$ is minimal for a (t,n)-threshold access structure if and only if $\#\mathcal{A} = t$.

Lemma 1. $\#\mathbf{K} \le \#\mathbf{S}$ *for any weakly perfect secret sharing scheme.*

Proof. Denote $\mathbf{K} = \{K_1, \ldots, K_\kappa\}$. We are going to consider the following two cases: every minimal $\mathcal{A} \in \Gamma$ satisfies $\#\mathcal{A} = 1$ and there exists a minimal $\mathcal{A}_0 \in \Gamma$ such that $\#\mathcal{A}_0 \ge 2$. The first case is trivial. For this case, let $M^*(i_1, 0) = K_1$ …, $M^*(i_\kappa, 0) = K_\kappa$. Since $K_1, \ldots, K_\kappa$ are mutually distinct, due to Condition (a), $M^*(i_1, 1)$, …, $M^*(i_\kappa, 1)$ must be mutually distinct. This proves that $\#\mathbf{S} \ge \#\mathbf{K}$. Consider the second case: there exists a minimal $\mathcal{A}_0 \in \Gamma$ such that $\#\mathcal{A}_0 \ge 2$. Let $\mathcal{A}_0 = \{P_{j_1}, \ldots, P_{j_\ell}\}$ where $j_1 < \cdots < j_\ell$. For fixed i_0th row of M^*, let $M^*(i_0, j_1) = s_{j_1}, \ldots, M^*(i_0, j_{\ell-1}) = s_{j_{\ell-1}}$. Since $\{P_{j_1}, \ldots, P_{j_{\ell-1}}\} \notin \Gamma$, according to Condition (b), for each K_r with $1 \le r \le \kappa$, there exists a row i_r of M^* such that $M^*(i_r, j_1) = s_{j_1}, \ldots, M^*(i_r, j_{\ell-1}) = s_{j_\ell - 1}$ and $M^*(i_r, 0) = K_r$, where $r = 1, \ldots, \kappa$. Since $M^*(i_1, 0) = K_1, \ldots, M^*(i_\kappa, 0) = K_\kappa$ are mutually distinct, due to Condition (a), $M^*(i_1, j_\ell), \ldots, M^*(i_\kappa, j_\ell)$ must be mutually distinct. This proves that $\mathbf{S}$ contains at least κ elements, i.e., $\#\mathbf{S} \ge \#\mathbf{K}$. □

A similar statement for perfect secret sharing appeared previously, for instance, in [3], that is, $\#\mathbf{K} \le \#\mathbf{S}$ for any perfect secret sharing scheme. Since any perfect secret sharing is a special weakly perfect secret sharing, Lemma 1 is more general. In particular, if the equality in Lemma 1 holds, i.e., $\#\mathbf{K} = \#\mathbf{S}$, the perfect secret sharing scheme is said to be ideal.

Definition 2. *A perfect secret sharing scheme is said to be* ideal *if* $\#\mathbf{K} = \#\mathbf{S}$*, where* $\mathbf{K}$ *and* $\mathbf{S}$ *denote the set of secrets and the set of shares respectively. Alternatively, a perfect threshold scheme is said to be* ideal *if the set of secrets and the set of shares have the same cardinality.*

Using the same approach as in the proof of Lemma 1, we can prove the following lemma.

Lemma 2. *Let M be an associated matrix M of an ideal secret sharing scheme with an access structure Γ. Let $\mathcal{A}_0 = \{P_{j_1}, \ldots, P_{j_\ell}\} \in \Gamma$, where $j_1 < \cdots < j_\ell$, be a minimal set. Then the submatrix of M, comprised of ℓ columns of M, indexed by $j_1, \ldots, j_\ell$, contains each row vector $(s_1, \ldots, s_\ell)$ where each $s_j \in \mathbf{S}$.*

In particular, we can formulate the following corollary.

Corollary 1. *Let M be an associated matrix of an ideal (t, n)-threshold scheme. Then a submatrix of M consisting of any t columns, contains all values of the vector $(s_1, \ldots, s_t)$ where each $s_j \in \mathbf{S}$.*

Let M^* be a defining matrix of an ideal (t, n)-threshold scheme. Set $\mathbf{S}^t = \{(s_1, \ldots, s_t) \mid s_1, \ldots, s_t \in \mathbf{S}\}$. Let $1 \le j_1 < \cdots < j_t \le n$ and M_1 be the $m \times t$ submatrix of M, comprised of the t columns indexed by $j_1, \ldots, j_t$. We now define a function, denoted by $\chi_{j_1,\ldots,j_t}$, from $\mathbf{S}^t$ to $\mathbf{K}$ as follows. According to Corollary 1, for any $(s_1, \ldots, s_t) \in \mathbf{S}^t$, there exists some i_0 with $1 \le i_0 \le m$ such that $M^*(i_0, j_1) = s_1, \ldots, M^*(i_0, j_t) = s_t$. Let $M^*(i_0, 0) = K$. Note that according to Condition (d), if there exists another i_1 $(1 \le i_1 \le m)$ such that $M^*(i_1, j_1) = s_1, \ldots, M^*(i_1, j_t) = s_t$, then $M^*(i_1, 0) = K$. Thus we can define K to be the image of $(s_1, \ldots, s_t)$ and write $K = \chi_{j_1,\ldots,j_t}(s_1, \ldots, s_t)$. We call $\chi_{j_1,\ldots,j_t}$ the *secret function* with respect to $j_1, \ldots, j_t$. Secret functions play an important role as a tool against cheating. This will be elaborated later.

5 Ideal Threshold Schemes from Orthogonal Arrays

An $m \times n$ matrix with entries from b-set $\mathbf{B}$ is called an *orthogonal array*, denoted by (m, n, b, t), if its any $m \times t$ submatrix contains all b^t possible row vectors precisely λ times. Clearly $m = \lambda b^t$. The parameters m, t and λ are called the *size*, the *strength* and the *index* of the orthogonal array, respectively, while n is called the number of *constraints* and b is called the number of *levels*.

Lemma 3. *An orthogonal array (m, n, b, t) with an index λ is an orthogonal array (m, n, b, ℓ) with an index $\lambda b^{t-\ell}$ where ℓ is any integer with $1 \le \ell \le t$.*

In particular, we can formulate the following corollary.

Corollary 2. *Each column of an orthogonal array (m, n, b, t) with entries from a b-set $\mathbf{B}$ contains each element of $\mathbf{B}$ precisely λb^{t-1} times, where λ is the index of the orthogonal array.*

This following statement is obvious.

Lemma 4. *Let O_1 be an $m \times n_1$ submatrix of an orthogonal array (m, n, b, t) with an index λ. If $n_1 \ge t$ then O_1 is an orthogonal array (m, n_1, b, t) with an index λ.*

Orthogonal arrays with index $\lambda = 1$, i.e, orthogonal arrays (b^t, n, b, t) have many interesting properties. The following bounds on the number of constraints for orthogonal arrays (b^t, n, b, t) was proved by Bush [4]:

Lemma 5. *For an orthogonal array (b^t, n, b, t),*

(i) if $t \le b$ then $n \le b + t - 1$ (b is even) or $n \le b + t - 2$ (b is odd and $t \ge 3$),
(ii) if $b \le t$, then $n \le t + 1$.

Theorem 1. *An orthogonal array $(b^t, n+1, b, t)$ with entries from a b-set* $\mathbf{B}$ *is a defining matrix of an ideal (t,n)-threshold scheme with* $\mathbf{K} = \mathbf{S} = \mathbf{B}$.

Proof. Let O be an orthogonal array $(b^t, n+1, b, t)$ with entries from b-set $\mathbf{B}$. We index the columns of O by $j = 0, 1, \ldots, n$ and index the rows of O by i, $1 \leq i \leq b^t$. We write $O(i,j)$ to denote the entry of O in the i row and the j column. We now construct a (t,n)-threshold with participants $P_1, \ldots, P_n$ as follows. For an ith row, let $O(i,0)$ be a secret, and $O(i,j)$ denote the share of participant of P_j, $j = 1, \ldots, n$. We next prove that this scheme satisfies Condition (c) and (d').

Let $\{P_{j_1}, \ldots, P_{j_\ell}\}$ be the set of currently active participants. For the case of $\ell \geq t$, if $O(i, j_1) = O(i', j_1), \ldots, O(i, j_\ell) = O(i', j_\ell)$, then it follows that $i = i'$, as the orthogonal array $(b^t, n+1, b, t)$ has index $\lambda = 1$. Thus Condition (c) is satisfied. For the case of $\ell < t$, let O_1 denote the $b^t \times (\ell + 1)$ submatrix of O, comprised of the $\ell + 1$ columns indexed by 0, j_1, …, j_ℓ. Note that $\ell + 1 \leq t$. Let $K, s_{j_1}, \ldots, s_{j_\ell} \in \mathbf{B}$. According to Lemma 3, O_1 contains the row vector $(K, s_{j_1} \ldots, s_{j_\ell})$ precisely $b^{t-\ell-1}$ times, where $b^{t-\ell-1}$ is independent to the choice of K. This proves (d'). Thus O is a defining matrix of a perfect (t,n)-threshold scheme. Finally, due to Corollary 2, we conclude that $\mathbf{K} = \mathbf{S} = \mathbf{B}$. Hence the scheme is ideal. □

6 Properties of Threshold Schemes from Orthogonal Arrays

The *Hamming distance* of two vectors $\mu = (a_1, \ldots, a_n)$ and $\nu = (b_1, \ldots, b_n)$, denoted by $dist(\mu, \nu)$, is the value of $\#\{j \mid a_j \neq b_j,\ 1 \leq j \leq n\}$.

Lemma 6. *Any two distinct row vectors of an orthogonal array (b^t, n, b, t) have a Hamming distance at least $n - t + 1$.*

Proof. Denote the orthogonal array by O. We prove the lemma by contradiction. Assume that there exist two rows of O, row L_i and L_j of O, satisfying $dist(L_i, L_j) \leq n - t$. Then L_i and L_j have at least t same corresponding coordinations. This contradicts the fact that the submatrix, comprised of any t columns, contains a row vector precisely once as O is an orthogonal array (b^t, n, b, t) with index $\lambda = 1$. Therefore we have proved the lemma. □

Consider (t,n)-threshold secret sharing whose defining matrix O is an orthogonal array $(b^t, n+1, b, t)$. Assume that the dealer chooses an i_0 row vector $(s_1, \ldots, s_n)$ of O and assigns $s_1, \ldots, s_n$ to participants $P_1, \ldots, P_n$, respectively. Let $\{P_{j_1}, \ldots, P_{j_\ell}\}$ for $t \leq \ell \leq n$ be a subset of active participants. Let O_1 be the $b^t \times \ell$ submatrix of O, containing ℓ columns indexed by $j_1, \ldots, j_\ell$. According to Lemma 4, O_1 is an orthogonal array (b^t, ℓ, b, t). Denote the ith row of O_1 by L_i. According to Lemma 6, any two distinct row vectors L_i and L_j of O_1 satisfy

$$dist(L_i, L_j) \geq \ell - t + 1 \qquad (2)$$

Clearly, the row i_0 of O_1 is $L_{i_0} = (s_{j_1}, \ldots, s_{j_\ell})$. Let there exist u cheaters, among the active participants $P_{j_1}, \ldots, P_{j_\ell}$, who submit modified shares to the combiner while the honest active participants submit correct shares to the combiner. Assume that the combiner receives the shares $s'_{j_1}, \ldots, s'_{j_\ell}$ sent by $P_{j_1}, \ldots, P_{j_\ell}$, where $s'_{j_i} = s_{j_i}$ if and only if P_{j_i} is honest. Write $L' = (s'_{j_1}, \ldots, s'_{j_\ell})$. Clearly, $dist(L', L_{i_0}) = u$.

We show that cheating can be checked when $1 \le u \le \ell - t$. We assume that the combiner (recovery algorithm) knows the defining matrix O and then O_1. Thus the combiner can calculate

$$d_m = \min\{dist(L', L_i) \mid 1 \le i \le b^t\}$$

Since $dist(L', L_{i_0}) = u$ and $1 \le u \le \ell - t$, it follows that $1 \le d_m \le \ell - t$. Although the combiner does not know L_{i_0}, from $1 \le d_m \le \ell - t$ and (2), he can conclude that $L' = (s'_{j_1}, \ldots, s'_{j_\ell})$ is not a row of O_1 and thus it is incorrect.

Furthermore we indicate that the correct shares can be found and the cheaters can be identified when $1 \le u \le \lfloor \frac{1}{2}(\ell - t) \rfloor$, where $\lfloor \frac{1}{2}(\ell - t) \rfloor$ denotes the greatest integer not larger than $\frac{1}{2}(\ell - t)$. The combiner can find a row L_{i_1} of O_1 such that $dist(L', L_{i_1}) = d_m$. Then L_{i_1} is identical with $L_{i_0} = (s_{j_1}, \ldots, s_{j_\ell})$. In fact $dist(L_{i_1}, L_{i_0}) \le dist(L_{i_1}, L') + dist(L', L_{i_0}) \le 2u \le \ell - t$. Since both L_{i_0} and L_i are rows of O_1, due to (2), we conclude that L_{i_1} is identical with $L_{i_0} = (s_{j_1}, \ldots, s_{j_\ell})$. Thus the correct shares have been found. Comparing L' and L_{i_0}, the combiner (recovery algorithm) can determine who are cheaters.

The above discussions uses basic facts of coding theory. The reader interested in more details is referred to any book on the subject.

7 Simple Construction

According to Theorem 1, the design of threshold schemes is equivalent to the construction of corresponding orthogonal arrays. In this section, we are interested in orthogonal arrays with elements in a finite field, or simply, orthogonal arrays *over* a finite field. Let $q = p^v$ where p is a prime number and v is a positive integer. We write $GF(q)$ or $GF(p^v)$ to denote the finite field of $q = p^v$ elements, and $GF(q)^n$ or $GF(p^v)^n$ to denote the vector space of n tuples of elements from $GF(q)$. Each vector $\alpha \in GF(q)^n$ can be expressed as $\alpha = (a_1, \ldots, a_n)$ where $a_1, \ldots, a_n \in GF(q)$. The integer $a_1 q^{n-1} + \cdots + a_{n-1} q + a_n$ is called the *integer representation* of vector $\alpha = (a_1, \ldots, a_n)$, where each a_j and the sum are regarded real-valued. Thus we can index all vectors in $GF(q)^n$:

$$\alpha_0, \alpha_1, \ldots, \alpha_{q^n - 1}$$

where j is the integer representation of α_j. A *function* f on $GF(q)^n$ is a mapping from $GF(q)^n$ to $GF(q)$. The function f can be expressed as $f(x)$ or $f(x_1, \ldots, x_n)$, where $x = (x_1, \ldots, x_n) \in GF(q)^n$. The *truth table* of f is the sequence $f(\alpha_0)$, $f(\alpha_1)$, ... , $f(\alpha_{q^n-1})$. If each element of $GF(q) = GF(p^v)$ appears in the truth table of f precisely q^{n-1} times then f is called *balanced*. If f can be expressed as $f(x_1, \ldots, x_n) = c + a_1 x_1 + \cdots + a_n x_n$ then f is called an *affine function*. In particular, the affine function f is called *linear* if $c = 0$. It is easy to see that non-constant affine functions are balanced.

For any integer t, n and prime power q with $1 \leq t \leq n+1 \leq q-1$, we next construct an orthogonal array $(q^t, n+1, q, t)$ over $GF(q)$. Since $n+1 \leq q-1$, we can collect $n+1$ nonzero elements of $GF(q)$: $\lambda_1, \ldots, \lambda_{n+1}$. For each λ_j, $1 \leq j \leq n+1$, define a vector $\beta_j = (1, \lambda_j, \ldots, \lambda_j^{t-1})$, $j = 1, \ldots, n+1$, and a liner function ψ_j on $GF(q)^t$ such that $\psi_j(x) = \langle \beta_j, x \rangle$ where $x = (x_1, \ldots, x_t) \in GF(q)^t$ and $\langle , \rangle$ denotes the inner product of two vectors. We now construct a $q^t \times (n+1)$ matrix O. We index the columns of O by $j = 0, 1. \ldots, n$, and define the j column vector of O to be the truth table of ψ_{j+1} where $j = 0, 1, \ldots, n$. According to the results given in [4], O is an orthogonal array $(q^t, n+1, q, t)$ over $GF(q)$. Therefore, by Theorem 1, O is a defining matrix of an ideal (t, n)-threshold scheme.

The above orthogonal arrays have a property as follows.

Lemma 7. *Let O be the orthogonal array $(q^t, n+1, q, t)$ over $GF(q)$, constructed previously in this section. Then for any fixed $1 \leq j_1 < \cdots < j_t \leq n$, $\chi_{j_1,\ldots,j_t}$ is a linear function.*

Proof. It is not hard to verify that any t vectors $\beta_{j_1}, \ldots, \beta_{j_t}$, where each β_j has been defined previously in this section, are linearly independent. Thus $\{\beta_{j_1}, \ldots, \beta_{j_t}\}$ is a basis of $GF(q)^t$ and thus the 0th column is a linear combination of the j_1th, ..., the j_tth columns. If we denote the jth columns of O by η_j, then $\eta_0 = c_1\eta_{j_1} + \cdots + c_t\eta_{j_t}$ where each $c_j \in GF(q)$. Thus $O(i,0) = c_1O(i,j_1) + \cdots + c_t0(i,j_t)$, $i = 1, \ldots, q^t$. By definition, $\chi_{j_1,\ldots,j_t}(s_1, \ldots, s_t) = c_1s_1 + \cdots + c_ts_t$, for any $s_1, \ldots, s_t \in GF(q)$. This proves the lemma. □

Using the same approach as shown by Tompa and Woll [7] for Shamir's scheme [6], due to Lemma 7, we can demonstrate that the Tompa-Woll attack works also for the scheme constructed above. For this reason, we will improve this construction in the next section.

8 Ideal Threshold Schemes with Nonlinear Secret Functions

We address the weakness discussed in the previous section by removing linearity from the orthogonal array $(q^t, n+1, q, t)$. Being more specific, we make sure that the 0th column (secret) is described by a nonlinear function of other columns (shares).

Theorem 2. *Let O be the orthogonal array $(q^t, n+1, q, t)$ over $GF(q)$, constructed in Section 7. We replace the 0th column by the truth table of function $\sigma(x) = \langle \beta_1, x \rangle^p$, where p is the characteristic of $GF(q)$, i.e., $q = p^v$. Denote the resulting matrix by O'. Then O' is also an orthogonal array $(q^t, n+1, q, t)$. Alternatively, we obtain an ideal (t, n)-threshold scheme with the defining matrix O'.*

Proof. Let O'_1 (O_1) be a $q^t \times t$ submatrix of O' (O), consisting of any t columns of O' (O), indexed by $j_1, \ldots, j_t$, where $0 \leq j_1 < \cdots < j_t \leq n$. Let $(a_1, a_2, \ldots, a_t)$ be a t-dimensional vector where each $a_j \in GF(q)$. There two cases to be considered: $j_1 \neq 0$ and $j_1 = 0$. For the first case: $j_1 \neq 0$, clearly $O'_1 = O_1$ is a submatrix of O. Thus $O'_1 = O_1$ contains $(a_1, a_2, \ldots, a_t)$ as a row vector precisely once. We next consider the second case: $j_1 = 0$. It is easy to verify that $c_1 = c_2$, where $c_1, c_2 \in GF(q)$, if and only if $c_1^p = c_2^p$. Thus, there exists an unique element $c \in GF(q)$ such that $c^p = a_1$. Recall that O is an orthogonal array $(q^t, n+1, q, t)$ with index $\lambda = 1$. Thus O_1 contains the row vector $(c, a_2, \ldots, a_t)$ precisely once. It follows that O'_1 contains the row vector $(a_1, a_2, \ldots, a_t)$ precisely once. Summarising the two cases, we have proved that O' is also an orthogonal array $(q^t, n+1, q, t)$. According to Theorem 1, we obtain an ideal (t, n)-threshold scheme with the defining matrix O'. □

Theorem 3. *Let O' be the orthogonal array $(q^t, n+1, q, t)$ in Theorem 2. For any $1 \leq j_1 < \cdots < j_t \leq n$, $\chi_{j_1,\ldots,j_t}$ is a nonlinear function.*

Proof. Recall that for each j with $1 \leq j \leq n$, the jth column of O' is the truth table of a linear function on $GF(q)^t$. On the other hand, the 0th column of O' is the truth table of the function $\sigma(x) = \langle \beta_1, x \rangle^p$, that contains nonlinear terms. Thus the 0th column of O' is not a linear combination of other columns. This proves that $\chi_{j_1,\ldots,j_t}$ is a nonlinear function. □

9 Constructions of Ideal Threshold Schemes

The construction in Section 7 demonstrates the existence of secret sharing based on orthogonal arrays. In this section, we show how to construct secret sharing from a known orthogonal array.

Theorem 4. *Let O be the orthogonal array (m, n, b, t) with elements from a b-set $\mathbf{B}$. For a permutation π on $\mathbf{B}$ and a uth column of O, we replace each entry c in the uth column by $\pi(c)$. Denote the resulting matrix by O'. Then O' is also an orthogonal array (m, n, b, t). Alternatively, we obtain an ideal threshold scheme based on the defining matrix O'.*

Proof. The proof is similar to the proof of Theorem 2. Let O'_1 (O_1) be an $m \times t$ submatrix of O' (O), consisting of the t columns of O' (O), indexed by $j_1, \ldots, j_t$, where $1 \leq j_1 < \cdots < j_t \leq n$. Let $(a_1, a_2, \ldots, a_t)$ be a t-dimensional vector where each $a_j \in GF(q)$. There two cases to be considered: $u \notin \{j_1, \ldots, j_t\}$ and $u \in \{j_1, \ldots, j_t\}$. For the first case: $u \notin \{j_1, \ldots, j_t\}$, clearly $O'_1 = O_1$ is a submatrix of O. Thus $O'_1 = O_1$ contains $(a_1, a_2, \ldots, a_t)$ as a row vector precisely once. We next consider the second case: $u \in \{j_1, \ldots, j_t\}$. Let $u = j_r$ and then assume that $j_1 < \cdots < j_{r-1} < j_r < j_{r+1} < \cdots < j_t$. Since π is a permutation on $\mathbf{B}$, there exists an unique element $c \in \mathbf{B}$ such that $\pi(c) = a_{j_r}$. Recall that O is an orthogonal array (m, n, b, t) with index $\lambda = 1$. Thus, O_1 contains the row vector $(a_1, \ldots, a_{j_{r-1}}, c, a_{j_{r+1}}, \ldots, a_t)$ precisely once. It follows that O'_1 contains the row vector $(a_1, \ldots, a_{j_{r-1}}, a_{j_r}, a_{j_{r+1}}, \ldots, a_t)$ precisely once. Summarising the two cases, we have proved that O' is also an orthogonal array (m, n, b, t). Alternatively, we obtain an ideal threshold scheme with the defining matrix O'. □

Repeatedly applying Theorem 4, we obtain more orthogonal arrays and more ideal threshold schemes. Moreover, the theorem gives ideal threshold schemes with different properties.

Theorem 5. *Let O be the orthogonal array $(q^t, n+1, q, t)$ over $GF(q)$, constructed in Section 7. Let a permutation π on $GF(q)$ satisfy $\pi(0) \neq 0$. For a uth $(1 \leq u \leq n)$ column of O, we replace each entry c in the uth column by $\pi(c)$, and replace the 0th column by the truth table of function $\sigma(x) = \langle \beta_1, x\rangle^p$, where $q = p^v$. Denote the resulting matrix by O'. Then*

(i) O' is also an orthogonal array (q^t, n, q, t). Alternatively, we obtain an ideal threshold scheme with the defining matrix O',
(ii) all the row vectors of the orthogonal array O' do not form a linear subspace of $GF(q)^n$,
(iii) for any $1 \leq j_1 < \cdots < j_t \leq n$, $\chi_{j_1,\ldots,j_t}$ is a nonlinear function.

Proof. According to Theorems 2 and 4, (i) is true. We denote the jth column of O (O') by η_j (η'_j). From the construction of O mentioned in Section 7, $O(0,u) = 0$. Thus $O'(0,u) = \pi(0) \neq 0$. This means that η'_u is not the true table of a linear function. We have proved (ii). We next prove (iii). There exist two cases to be considered: $u \notin \{j_1,\ldots,j_t\}$ and $u \in \{j_1,\ldots,j_t\}$. In the first case: $u \notin \{j_1,\ldots,j_t\}$. According to the same arguments as in the proof of Theorem 3, (iii) is true in the first case. We consider the second case: $u \in \{j_1,\ldots,j_t\}$. Let $u = j_r$. We prove (iii) by contradiction. Assume that $\chi_{j_1,\ldots,j_t}$ is a linear function. By definition,

$$\eta'_0 = c_1\eta'_{j_1} + \cdots + c_{r-1}\eta'_{j_{r-1}} + c_r\eta'_{j_r} + c_{r+1}\eta'_{j_{r+1}} + \cdots + c_t\eta'_{j_t}$$

for some $c_1,\ldots,c_t \in GF(q)$. Since $\eta'_j = \eta_j$ for $j \neq 0, j_r$, we have

$$\eta'_0 = c_1\eta_{j_1} + \cdots + c_{r-1}\eta_{j_{r-1}} + c_r\eta'_r + c_{r+1}\eta_{j_{r+1}} + \cdots + c_t\eta_{j_t} \tag{3}$$

From the proof of Theorem 3, we know that η'_0 is not a linear combination of $\eta_{j_1}, \ldots, \eta_{j_{r-1}}, \eta_{j_{r+1}}, \ldots, \eta_{j_t}$. Thus we conclude that $c_r \neq 0$. On the other hand, $O'(0,0) = 0$, $O(0,j_1) = 0$, …, $O(0,j_{r-1}) = 0$, $O(0,j_{r+1}) = 0$, …, $O(0,j_t) = 0$ but $O'(0,j_r) \neq 0$. This means that (3) does not hold. The contradiction proves (iii) in the second case. □

It is easy to see that all row vectors of the orthogonal array $(q^t, n+1, q, t)$, constructed in Section 7, form a linear subspace. Usually, this is not a desirable property from a security point of view as the corresponding secret sharing may be subject to the Tompa-Woll attack. In contrast to the construction in Section 7, the construction in Theorem 5 provides secret sharing that is resistant against cheating.

10 Conclusions

In this work we have applied orthogonal arrays to construct threshold schemes and have shown that all these schemes are not only perfect but also ideal. We have indicated that such ideal threshold schemes have an ability to detect cheating and also, can identify cheaters and recover correct shares. Besides cheating detection and identification, we have also shown that the secret functions must be nonlinear to prevent cheating using the Tompa-Woll attack.

Acknowledgement

The work was in part supported by Australian Research Council grant A00103078.

References

1. G. R. Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313–317. AFIPS, 1979.
2. E. F. Brickell and D. M. Davenport. On the Classification of Ideal Secret Sharing Schemes. J. Cryptology, 4: 123 - 134, 1991.
3. E. F. Brickell and D.R. Stinson. Some Improved Bounds on Information Rate of Perfect Sharing Schemes J. Cryptology, 5: 153 - 166, 1992.
4. K. A. Bush. Orthogonal arrays of index unity. In *Annals of Mathematical Statistics.* 23, 426-434.
5. M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *Proceedings IEEE Globecom '87*, pages 99–102. IEEE, 1987.
6. A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.
7. M. Tompa and H. Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(2):133–138, 1988.

Cryptanalysis of the Reduced-Round RC6

Atsuko Miyaji[1] and Masao Nonaka[2]

[1] School of Information Science, Japan Advanced Institute of Science and Technology
miyaji@jaist.ac.jp
[2] Matsushita Electric Industrial Co., LTD.
nonaka@isl.mei.co.jp

Abstract. In this paper, we propose the efficient and feasible key recovery algorithm against the reduced-round RC6 without whitening, called RC6W. Our attack applies to a rather large number of rounds. RC6W with r rounds can be broken in a success probability of 90% by using $2^{8.1r-13.8}$ plaintexts. Therefore, our attack can break RC6W with 17 rounds by using $2^{123.9}$ plaintexts in a probability of 90%.

1 Introduction

RC6-$w/r/b$[13] is a fast software-oriented block cipher, which is constructed by only simple arithmetic such as a multiplication, an addition, a bit-wise exclusive-or(XOR), and a data dependent rotation. RC6-$w/r/b$ means that four w-bit-word plaintexts are encrypted by r rounds with b byte keys. We denote RC6-32/r/16 by RC6, which is submitted as a candidate for NESSIE[12], and recently has been selected to proceed the next stage. RC6 is the next version of RC5[14]. RC5 also includes a data dependent rotation, which is much efficiently improved in RC6 in such a way that it is determined by all 32 bits of both data and subkey but not 5 bits. Such an efficient improvement makes RC6 much secure because it is difficult to handle the rotation by choosing specific plaintexts. Compared with various attacks against RC5[1,2,4,5,6,7,11], any key recovery algorithm against RC6[3,2,8] requires much memory and work even in the case of low round. Multiple linear cryptanalysis is applied to RC6-32/r/32[16], but it has not been applied to RC6-32/r/16.

Correlation attack makes use of correlations between an input and an output, which is measured by the χ^2 test: the specific rotation in RC6 is considered to cause the correlations between the corresponding two 10-bit integer values. Correlation attack consists of two parts, the distinguishing algorithm and the key recovery algorithm. The distinguishing algorithm has only to handle plaintexts in such a way that the χ^2-value of a part of the ciphertext becomes a significantly higher value. The key recovery algorithm has to rule out all false keys, and single out exactly a correct key by using the χ^2-value. Up to the present, only distinguishing algorithm has been investigated[8,4]. That is, only the high χ^2-value is focused, which is experimentally computed on the average of keys. In [8], correlation attacks are applied to RC6 to recover subkeys from the 1st subkey to the final subkey. Their key recovery algorithm is based on the next idea: the χ^2-value is significantly high if B_0 or D_0 of a plaintext (A_0, B_0, C_0, D_0) introduces zero rotation in the 1st round, and $lsb_5(A_0)$ and $lsb_5(C_0)$ is fixed. It exactly works well as a distinguishing algorithm, but, as a key recovery algorithm, it is unlikely that it rules out all 32-bit false keys well. Because fixing the first round rotation to zero is just fixing the 5-bit information amount on the first subkey but not all its 32bits. In fact, for a plaintext, there are 2^{27} first subkeys that lead to the zero rotation. Furthermore, unfortunately, their key recovery algorithm has not been executed yet although their distinguishing algorithm has been implemented. Because their algorithm is forced to recover all 32 bits of the first subkey, and thus it requires $2^{62.2}$ works with 2^{42} memory even in the case of RC6 with 5 rounds. In a realistic sense, it would be infeasible to employ such an algorithm on a modern computer. This is why their key recovery algorithm is estimated by only using the results of distinguishing algorithm. We also note that the number of available plaintexts for each key is 2^{118}. In [11], a correlation attack against RC5 is proposed. Their algorithm can recover every four bit of subkey in the final round with the high probability by using a rather low χ^2-value. They also reported that an algorithm applying [8] to RC5 cannot recover subkeys well although the χ^2-value is extremely high. Their results indicate that any bit of subkeys can be recovered, and that a good distinguishing algorithm does not necessarily work as a good key recovery algorithm.

In this paper, we focus on RC6 without whitening, which is called RC6W in this paper. We propose the feasible key recovery algorithm for the reduced-round RC6W for the first time. We improve the distinguishing algorithm in such a way that the χ^2-values for outputs become significantly high with less constrain of plaintexts, and then improve key recovery algorithm in such a way that the variance of χ^2-value is reduced. As for the distinguishing algorithm, we investigate how outputs after r rounds, both A_{r+1} and C_{r+1}, depend on a chosen plaintext, and find experimentally the following feature of RC6: the χ^2-values for the concatenation of $lsb_5(A_{r+1})$ and $lsb_5(C_{r+1})$ become significantly high if both the least significant 5 bits of the first and third words before addition to each 1st-round subkey are just fixed. This means that we can use any plaintext by classifying them into groups with the same condition, and thus, the number of available plaintext is 2^{128}. As for the key recovery algorithm, we also direct our attention to the variance of χ^2-value in addition to the above results of key distinguishing algorithm. We compute the χ^2-value not flatly for all plaintext but for plaintexts in each group, and then compute the average among these χ^2-value. As a result, the variance of χ^2-value is reduced. The main points of our feasible key recovery algorithm are as follows:

1. Use any plaintext by classifying it into groups;
2. Compute the χ^2-value of an output for plaintexts in each group, and then compute the average among these χ^2-value.

We also present three key recovery algorithms, which reflect the importance of the variance of the χ^2-value. By employing our best attack, RC6W with 5 rounds can be broken within 20 minutes on PPC 604e/332MHz by using 2^{27} plaintexts and 2^{26} memory. RC6W with r rounds can be broken with a success probability of 90% by using $2^{8.1r-13.8}$ plaintexts, and thus RC6W with 17 rounds can be broken in a probability of 90% by using $2^{123.9}$ plaintexts and feasible memory size of 2^{26}, faster than an exhaustive key search. We also investigate a two-register-version RC6[13], called RC6-64 in this paper. We denote RC6-64 without whitening by RC6-64W. RC6-64 is oriented to 64-bit architecture, whose plaintexts consists of two 64-bit words, and whose size of subkeys is 64 bits. So the security level of one round in RC6-64, the size of subkeys, is equal to that in RC6-32, which has two 32-bit subkeys in one round. The round function of RC6-64 is almost the same structure as that of RC6. It is very useful to discuss the difference of each security of round function. By applying our attack to RC6-64W with r rounds, it can be broken in a success probability of 90% by using $2^{5.0r-8.2}$ plaintexts, and thus RC6-64W with 27 rounds can be broken by using $2^{126.8}$ plaintexts in a probability of 90%. The weakpoint of RC5 is thought to a data dependent rotation, which is defined by only 5-bit subkey and data, but not the data structure of two words. Although the weakness of data dependent rotation is improved in both RC6 and RC6-64, RC6-64 is much weaker than RC6. From our results, we see that the data structure of RC6, 4-word plaintexts, also makes the security high.

This paper is organized as follows. Section 2 summarizes some notations and definitions in this paper. Section 3 describes experimental results of χ^2-test of RC6. Section 4 presents two chosen plaintext algorithm, Algorithm 2 and 3, and one known plaintext algorithm, Algorithm 4. Section 5 applies Algorithm 4 for RC6-64W, and discusses the difference between RC6 and RC6-64W from a security point of view.

2 Preliminary

This section denotes RC6 algorithm and some experimental remarks after defining the following notations:

$+, \boxplus\ (-, \boxminus)$: an addition (subtraction) mod 2^{32}; $\oplus$: a bit-wise exclusive OR;
$a \lll (\ggg) b$: a cyclic rotation of a to the left(right) by b bits;
S_i : the i-th subkey; r : the number of (full)rounds;
$lsb_n(X)$: the least significant n bits of X; X^i : the i-th bit of X;
$X^{[i,j]}$: from the i-th bit to the j-th bit of X $(i > j)$;
$f(X)$: $X \times (2X+1)$; $F(X)$: $f(X) \pmod{2^{32}} \lll 5$.

Algorithm 1 (Encryption with RC6)

```
1. $A_1 = A_0$; $B_1 = B_0 + S_0$; $C_1 = C_0$; $D_1 = D_0 + S_1$;
2. for $i = 1$ to $r$ do: $t = F(B_i)$; $u = F(D_i)$; $A_{i+1} = B_i$;
       $B_{i+1} = ((C_i \oplus u) \lll t) + S_{2i+1}$; $C_{i+1} = D_i$; $D_{i+1} = ((A_i \oplus t) \lll u) + S_{2i}$;
3. $A_{r+2} = A_{r+1} + S_{2r+2}$; $B_{r+2} = B_{r+1}$; $C_{r+2} = C_{r+1} + S_{2r+3}$; $D_{r+2} = D_{r+1}$.
```

The part 1 or 3 is called to pre-whitening or post-whitening, respectively. We call the version of RC6 without pre- or post-whitening to RC6W or RC6 without whitening.

Table 1. χ^2-distribution with each degree of freedom

Level	0.50	0.60	0.70	0.80	0.90	0.95	0.99
31 degree of freedom	30.34	32.35	34.60	37.36	41.42	44.99	52.19
63 degree of freedom	62.33	65.20	68.37	72.20	77.75	82.53	92.01
255 degree of freedom	254.33	260.09	266.34	273.79	284.34	293.25	310.46
1023 degree of freedom	1022.33	1033.83	1046.23	1060.86	1081.38	1098.52	1131.16

We make use of the χ^2-tests for distinguishing a random sequence from non-random sequence [6,8,9]. Let $X = X_0, ..., X_{n-1}$ be a sequence with $\forall X_i \in \{a_0, \cdots, a_{m-1}\}$. Let $N_{a_j}(X)$ be the number of X_i which equals a_j. The χ^2-statistic of X, $\chi^2(X)$, estimates the difference between X and the uniform distribution as follows: $\chi^2(X) = \frac{m}{n} \sum_{i=0}^{m-1} \left(N_{a_i}(X) - \frac{n}{m}\right)^2$. Table 1 presents each threshold for 31, 63, 255, 1023 degrees of freedom. For example, (level, χ^2)=(0.95, 44.99) for 31 degrees in Table 1 means that the value of χ^2-statistic exceeds 44.99 in the probability of 5% if the observation X is uniform. In this paper, we uses these four degree of freedom. For preciseness, we often discuss the χ^2-statistic for any degree by the level. We set the level to 0.95 in order to distinguish a sequence X from a random sequence.

In our experiments, all plaintexts are generated by using m-sequence[10]. For example, Algorithm 2, 3, or 4 uses 108-, 113- or 128-bit random number generated by m-sequence, respectively. The platforms are IBM RS/6000 SP (PPC 604e/332MHz $\times$ 256) with memory of 32 GB.

3 Distinguishing Algorithm for RC6

In this section, we investigate how to lead to much stronger biases with less constraint of plaintexts.

3.1 χ^2-Statistic of RC6

We conduct the following experiments in order to find conditions of less constraint of plaintexts.

Test 1: χ^2-test on $lsb_5(A_{r+1})||lsb_5(C_{r+1})$ in the case which both B_0 and D_0 introduce zero rotation in the 1st round, $lsb_5(A_0) = 0$, and $lsb_5(C_0) = 0$.
Test 2: χ^2-test on $lsb_5(A_{r+1})||lsb_5(C_{r+1})$ in the case which both B_0 and D_0 introduce zero rotation in the 1st round, $lsb_5(A_0) = 0, ..., 31$, and $lsb_5(C_0) = 0$.
Test 3: χ^2-test on $lsb_n(A_{r+1})||lsb_n(C_{r+1})$ for $n = 3, 4, 5$ in the case which both $lsb_5(A_0)$ and $lsb_5(C_0)$ are set to 0, and both B_0 and D_0 introduce zero rotation in the 1st round.
Test 4: χ^2-test on (any consecutive 5 bits of A_{r+1}) $||lsb_5(C_{r+1})$ in the case which both $lsb_5(A_0)$ and $lsb_5(C_0)$ are set to 0, and both B_0 and D_0 introduce zero rotation in the 1st round.

Table 2. The χ^2-value on $lsb_5(A_{r+1})||lsb_5(C_{r+1})$ of RC6 in Test 1(in 100 keys)

4 rounds									
# texts	2^{12}			2^{13}			2^{14}		
The χ^2-value	Average	Level	Variance	Average	Level	Variance	Average	Level	Variance
	1045.450	0.694	1774.828	1076.568	0.881	2177.806	1126.800	0.987	2448.999

6 rounds									
# texts	2^{28}			2^{29}			2^{30}		
The χ^2-value	Average	Level	Variance	Average	Level	Variance	Average	Level	Variance
	1041.933	0.667	2098.079	1060.985	0.801	2263.724	1095.914	0.944	2942.704

The condition in Test 1 is the same with that in [8]. The conditions in other tests are eased than Test 1. We observe whether the almost same effect as Test 1 is expected with the eased conditions or not.

We discuss the difference between Test 1 and 2. In the first round, each of A_1 and C_1 is added to each round key, and thus neither $lsb_5(A_1)$ nor $lsb_5(C_1)$ is zero in the final stage of the first round even if plaintexts are chosen with the condition of Test 1. Therefore, the same effect as $lsb_5(A_0), lsb_5(C_0) = 0$ is expected if only $lsb_5(A_0)$ and $lsb_5(C_0)$ is just fixed. Test 2 examines the hypothesis. Table 2 shows the implementation results in the case of $r = 4, 6$ of Test 1. We compute the χ^2-value on $lsb_5(A_{r+1})||lsb_5(C_{r+1})$ on the average of 100 keys, and the level and the variance. The variance will be discussed in the following sections. The number of available plaintexts in Test 1 is 2^{108}. The experimental results of Test 2 are presented in Figure 1. The horizontal line corresponds to the fixed value of $lsb_5(A_0)$ and the vertical line corresponds to the level of the χ^2-value on $lsb_5(A_5)||lsb_5(C_5)$ for each number of plaintexts. From Figure 1, we see that any $lsb_5(A_0)$ can be distinguished from a random sequence in almost the same way as $lsb_5(A_0) = 0$. The same discussion also holds in the case of $lsb_5(C_0)$. To sum up, we do not have to set $lsb_5(A_0) = lsb_5(C_0) = 0$ in order to increase the χ^2-value. We can use plaintexts with any (A_0, C_0) by just classifying it to each $lsb_5(C_0)$ and $lsb_5(A_0)$, and thus the number of available plaintexts is 2^{118}.

Test 3 examines whether outputs with any bit-size n of $lsb_n(A_{r+1})||lsb_n(C_{r+1})$ lead also highly nonuniform distribution or not. Our key recovery algorithm shown in Section 4 and 4.3 can set the size of recovered key flexibly. Therefore if the nonuniform distribution of $lsb_n(A_{r+1})||lsb_n(C_{r+1})$ for $n \neq 5$ also holds, then our algorithm can work according to the memory capacity of machine. The experimental results of Test 3 in the case of 4, 6 rounds are presented in Table 3. From the experimental results, we see that the larger n is, the higher the nonuniform distribution of $lsb_n(A_{r+1})||lsb_n(C_{r+1})$ is, and that the nonuniform distribution of $lsb_n(A_{r+1})||lsb_n(C_{r+1})$ for $n = 3, 4$ is also observed in the same way as $n = 5$. Since we use the χ^2-value on $lsb_3(A_{r+1})||lsb_3(C_{r+1})$ in Section 4, other experimental results in the case of $lsb_3(A_{r+1})||lsb_3(C_{r+1})$ are shown in Table 4.

Test 4 computes the χ^2-value in (any consecutive 5 bits of $A_{r+1})||lsb_5(C_{r+1})$. Figure 2 shows the experimental results in the case of 4 rounds. The horizon-

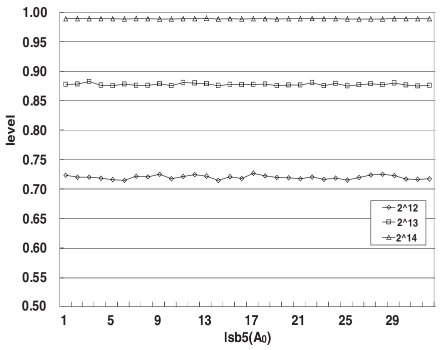


Fig. 1. Level of the χ^2-value of RC6 in Test2 (in 10^4 keys)

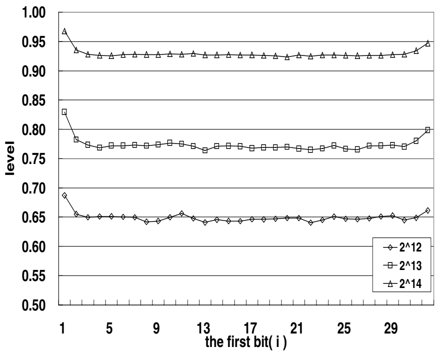


Fig. 2. Level of the χ^2-value of RC6 in Test 4 (in 10^4 keys)

tal line corresponds to the first bit of consecutive 5 bits of A_5, and each plot presents the level of χ^2-value in the case of each consecutive 5 bits for each number of plaintexts. For example, the case of $i = 1$, or $i = 32$ corresponds to $A_5^{[5,1]}$, or $\{A_5^{32}, A_5^{[4,1]}\}$. From Figure 2, we see that (any consecutive five bits of A_5)$||lsb_5(C_5)$ can be distinguished from a random sequence in almost the same way as $lsb_5(A_5)||lsb_5(C_5)$.

3.2 χ^2-Statistic of RC6 without Pre-whitening

We focus attention on RC6 without pre-whitening, called RC6WP. In Test 2, we have seen that the χ^2-value on $lsb_5(A_{r+1})||lsb_5(C_{r+1})$ becomes significantly high if both B_0 and D_0 introduce zero rotation in the 1st round, and both $lsb_5(A_0)$ and $lsb_5(C_0)$ are fixed. That is, in Test 2, both $lsb_5((A_0 \oplus F(B_0+S_0)) \lll F(D_0+S_1))$ and $lsb_5((C_0 \oplus F(D_0 + S_1)) \lll F(B_0 + S_0))$ are fixed. Therefore, in the case of RC6WP, the same effect as Test 2 is expected if only both $lsb_5((A_0 \oplus F(B_0)) \lll F(D_0))$ and $lsb_5((C_0 \oplus F(D_0)) \lll F(B_0))$ are fixed. To observe this, we do the next experiments.

Test 5: χ^2-test on $lsb_5(A_{r+1})||lsb_5(C_{r+1})$ of RC6WP with $lsb_5((C_0 \oplus F(D_0)) \lll F(B_0)) = 0$ and $lsb_5((A_0 \oplus F(B_0)) \lll F(D_0)) = 0$.
Test 6: χ^2-test on $lsb_5(A_{r+1})||lsb_5(C_{r+1})$ of RC6WP with $lsb_5((C_0 \oplus F(D_0)) \lll F(B_0)) = 0$ and $lsb_5((A_0 \oplus F(B_0)) \lll F(D_0)) = 0, .., 31$.

Table 5 shows the result of Test 5 in the case of 4 rounds. Compared with Table 2, we see that the effect of Test 5 is better than that of Test 1. Figure 3 presents the experimental results of Test 6: the horizontal line corresponds to the fixed value of $lsb_5((A_0 \oplus F(B_0)) \lll F(D_0))$ and the vertical line corresponds to the χ^2-value of $lsb_5(A_5)||lsb_5(C_5)$ for each number of plaintexts. From Figure 1, we see that any $lsb_5((A_0 \oplus F(B_0)) \lll F(D_0))$ can be distinguished from a random sequence in almost the same way as $lsb_5((A_0 \oplus F(B_0)) \lll F(D_0)) = 0$. The same discussion also holds in the case of $lsb_5((C_0 \oplus F(D_0)) \lll F(B_0)) = 0$. In the case of analysis of RC6WP, we can handle plaintexts by controlling $lsb_5((A_0 \oplus$

Table 3. The χ^2-value on $lsb_n(A_{r+1})||lsb_n(C_{r+1})$ of RC6 in Test 3(in 100 keys)

4 rounds									
# texts	2^{12}			2^{13}			2^{14}		
χ^2-value	Average	Level	Variance	Average	Level	Variance	Average	Level	Variance
$n = 3$	66.275	0.635	140.251	69.518	0.733	155.518	81.111	0.938	244.195
$n = 4$	268.910	0.737	493.753	277.883	0.845	618.303	301.961	0.977	679.494
$n = 5$	1045.450	0.694	1774.828	1076.568	0.881	2177.806	1126.800	0.987	2448.973

6 rounds									
# texts	2^{29}			2^{30}			2^{31}		
χ^2-value	Average	Level	Variance	Average	Level	Variance	Average	Level	Variance
$n = 3$	71.804	0.791	203.645	76.572	0.883	209.564	88.474	0.981	270.062
$n = 4$	273.571	0.797	580.289	290.854	0.939	699.839	323.876	0.998	1049.104
$n = 5$	1060.985	0.801	2263.680	1095.913	0.944	2942.691	1173.418	0.999	3270.362

Table 4. The χ^2-value on $lsb_3(A_5)||lsb_3(C_5)$ of RC6 in Test 3(in 10^5 keys)

# texts	2^7			2^8			2^9		
χ^2-value	Average	Level	Variance	Average	Level	Variance	Average	Level	Variance
	63.174	0.530	126.426	63.241	0.532	126.612	63.395	0.538	126.645

# texts	2^{10}			2^{11}		
χ^2-value	Average	Level	Variance	Average	Level	Variance
	63.820	0.553	130.434	64.655	0.581	131.970

$F(B_0)) \lll F(D_0))$. We can use any plaintext to analysis for RC6WP by just classifying it into each $lsb_5((A_0 \oplus F(B_0)) \lll F(D_0))$ and $lsb_5((C_0 \oplus F(D_0)) \lll F(B_0))$, and thus the number of available plaintexts is 2^{128}.

3.3 The Variance of χ^2-Distribution

We have seen from the experimental results that high correlations between an input and an output of RC6 are observed if both inputs and outputs are chosen appropriately. Correlation attack makes use of the correlation: if we choose a correct key, then high correlations between an input and an output of RC6 would be observed; but if we choose a false key, then high correlations between an input and an output of RC6 would not be observed. In distinguishing algorithm, the χ^2-value is computed on the average of keys, and thus only the conditions, of which the average of χ^2-value is high, are discussed. However, each experimental results show that variance of distribution of the χ^2-value can not be negligible in the case of correct keys. Generally, for a normally distributed X with the average μ, and the variance σ^2, the probability that the data exists in $\{\mu - \sigma \le X \le \mu + \sigma\}$, $\Pr(\mu - \sigma \le X \le \mu + \sigma)$, satisfies $\Pr(\mu - \sigma \le X \le \mu + \sigma) = 0.68$. Therefore, if the variance would not be reduced, then we could not rule out all false keys, and single out exactly a correct key. In the following sections, we will design key recovery algorithms in such a way that the variance of χ^2-distribution is reduced.

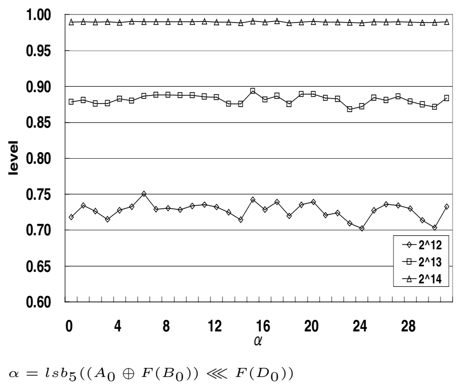


Fig. 3. The χ^2-value of RC6WP in Test 6 (in 10^4 keys)

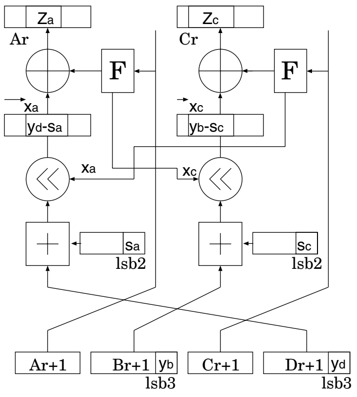


Fig. 4. Outline of Algorithm 2

3.4 Estimation

In the following sections, we will show key recovery algorithms, based on χ^2-test. We actually implement theses key recovery algorithms against RC6W with 5 rounds, and evaluate the χ^2-value necessary for key recovery against RC6W with 5 rounds exactly. For the discussion against RC6W with more rounds, we use the same method as [8] to estimate the complexities of key recovery algorithms: we estimate *slope*, that is, how many plaintexts are required to get similar values in a χ^2-test on $r+1$ rounds compared with r rounds. Our algorithms, applied to RC6W, may make use of χ^2-test of RC6 to estimate the slope: as for the post-whitening, the χ^2-value without post-whitening is the same as that with post-whitening; and as for the pre-whitening, the condition without pre-whitening is the same as that of which B_0 and D_0 introduce zero rotation in the 1st round of RC6. The condition of χ^2-test of three key recovery algorithms are classified into two cases: Condition 1(Algorithm 2 and 3) and Condition 2(Algorithm 4).

Condition 1 The χ^2-test on $lsb_3(A_{r+1})||lsb_3(C_{r+1})$ of RC6 in the case which both B_0 and D_0 introduce zero rotation in the 1st round, $lsb_5(A_0) = 0$, and $lsb_5(C_0) = 0$.

Condition 2 The χ^2-test on $lsb_3(A_{r+1})||lsb_3(C_{r+1})$ of RC6 in the case which both B_0 and D_0 introduce zero rotation in the 1st round, $lsb_3(A_0) = 0$, and $lsb_3(C_0) = 0$.

Condition 1 is the same with the case of $n = 3$ in Test 3. Table 6 shows the precise experimental results in Condition 1 and 2. It presents the number of plaintexts required for the χ^2-value with each level, 0.70, 0.75, 0.80, 0.90, and 0.95, which is calculated to the first decimal place. From Table 6, we can estimate that to get similar values in a χ^2-test on $r+1$ rounds compared r rounds requires a factor of $2^{8.1}$ additional plaintexts in both Condition 1 and 2.

Table 5. The χ^2-value on $lsb_5(A_5)||lsb_5(C_5)$ of RC6WP in Test 5(in 100 keys)

# texts	2^{12}			2^{13}			2^{14}		
χ^2-value	Average	Level	Variance	Average	Level	Variance	Average	Level	Variance
	1054.720	0.761	2653.532	1083.073	0.906	2634.250	1137.702	0.993	2504.252

Table 6. $log_2\#$(texts) required for the χ^2-value of RC6 with each level

	4 rounds					4 rounds				
Level	0.70	0.75	0.80	0.90	0.95	0.70	0.75	0.80	0.90	0.95
Condition 1	12.5	12.9	13.1	13.8	14.2	28.3	28.6	29.2	30.2	30.7
Condition 2	14.9	15.3	15.6	16.1	16.6	30.8	31.1	31.6	32.5	32.8

4 Key Recovery Algorithms against RC6W

In this section, we present two chosen-plaintext key recovery algorithms and one known-plaintex key recovery algorithm against RC6W.

4.1 Algorithm 2

Algorithm 2 is designed by making use of the results in Section 3 as follows:
1. The χ^2-statistic are not measured on a fixed part of $A_{r+1}||C_{r+1}$ (**Test 4**);
2. The degree of χ^2-statistic is flexibly set to 63 in such a way that Algorithm 2 is feasible, that is, compute the χ^2-statistic on 6 bits of $A_{r+1}||C_{r+1}$ (**Test 3**);
3. The χ^2-value is computed on $z_a||z_c$, to which $lsb_3(B_{r+1})||lsb_3(D_{r+1})$ is exactly decrypted by 1 round (see Figure 4);
4. The χ^2-value is computed on each decrypted data $z_a||z_c$, which is classified into 64 cases according to each r-th round rotation number.

Algorithm 2

```
Recover $lsb_2(S_{2r})$ and $lsb_2(S_{2r+1})$ of RC6W.
Set $(lsb_3(B_{r+1}), lsb_3(D_{r+1})) = (y_b, y_d)$, $(lsb_2(S_{2r}), lsb_2(S_{2r+1})) = (s_a, s_c)$,
and $(lsb_5(F(A_{r+1})), lsb_5(F(C_{r+1}))) = (x_c, x_a)$.
1. Encrypt $(A_0, B_0, C_0, D_0)$
   with $(lsb_5(A_0), lsb_5(C_0), lsb_5(F(B_0)), lsb_5(F(D_0))) = (0,0,0,0)$.
2. For each $s_a, s_c = 0,1,2,3$, set $s = s_a||s_c$, $S^3_{2r}, S^3_{2r+1} = 0$, and decrypt $y_d||y_b$
   with key $(S^3_{2r}||s_a, S^3_{2r+1}||s_c)$ by 1 round, using the $r$-th round rotation
   amount $x_a$ and $x_c$. The decryptions of $y_d||y_b$ are set to $z_a||z_c = z$.
3. For each value $s$, $x_a$, $x_c$, and $z$, increment each array $count[s][x_a][x_c][z]$.
4. For each $s$, $x_a$, and $x_c$, compute $\chi^2[s][x_a][x_c]$.
5. Compute the average $ave[s]$ of $\{\chi^2[s][x_a][x_c]\}$ for each $s$, and output $s$
   with the highest $ave[s]$ as $lsb_2(S_{2r})||lsb_2(S_{2r+1})$.
```

Algorithm 2 computes the χ^2-value on z, to which y is decrypted by the final round subkey. Since the χ^2-value on the decryption z by using each key, $lsb_3(S_{2r})||lsb_3(S_{2r+1}) = 1||s_a||1||s_c, 1||s_a||0||s_c, 0||s_a||1||s_c, 0||s_a||0||s_c$ are coincident each other[11], we may decrypt y by setting $S^3_{2r}, S^3_{2r+1} = 0$ temporarily. Algorithm 2 shown the above works as 6-bit examination and 4-bit estimation,

Table 7. Success probability and the χ^2-value of Algorithm 2 (in 100 keys)

#texts	#keys	χ^2-value(63 degree)		
		Average	Level	Variance
2^{17}	12	63.106	0.527	0.165
2^{18}	8	63.076	0.526	0.122
2^{19}	16	63.216	0.531	0.109
2^{20}	32	63.492	0.541	0.107
2^{21}	71	64.049	0.561	0.102
2^{22}	99	65.119	0.597	0.133
2^{23}	100	67.321	0.668	0.218

Table 8. $\log_2$(#texts) required for recovering a key in Algorithm 2(in 100 keys)

	90%	70%	30%
$\log_2(\#text)$	21.4	21.0	20.0

but it can work flexibly as $2n$-bit examination and $2(n-1)$-bit estimation for $n = 3, 4, 5$ according to the capacity of memory. We can recover other bits of round keys S_{2r}, and S_{2r+1} by repeating Algorithm 2 sequentially. Apparently, the number of available plaintexts is 2^{108}.

Table 7 shows the results for RC6W with 5 rounds: the success probability among 100 trials, the average of χ^2-value of recovered keys, the level, and the variance. Let us compare the results in Algorithm 2 with Table 4. In Algorithm 2, the χ^2-value is computed on each group, classified by the rotation number in the final round. Since all plaintexts in our experiments are randomly generated by m-sequences, plaintexts are roughly estimated to be uniformly distributed to each group. Therefore, the χ^2-test is computed by using $1/2^{10}$ times the number of plaintexts in Table 7. The χ^2-test of using $2^{20} - 2^{23}$ plaintexts in Algorithm 2 corresponds to that of $2^{10} - 2^{13}$ in the case of $n = 3$ of Test 3. In a sense, Algorithm 2 computes the χ^2-value for sample mean, which keeps the average of χ^2-value but reduce the variance of χ^2-value from statistical fact. Comparing Table 7 with Table 3 and 4, we see that the variance of χ^2-value in Algorithm 2 is about $1/2^{10}$ as much as that in the corresponding Test3, and that the average of χ^2-value in Algorithm 2 is almost the same as that in the corresponding Test3. Algorithm 2 can recover a key with rather low level by reducing the variance of χ^2-value.

More precise experimental results are shown in Table 8. All experiments are calculated to the first decimal place. From Table 8, the number of plaintexts required for recovering a key in r rounds with the success probability of 90%, $log_2(\#text)$, is estimated to $log_2(\#text) = 8.1r - 19.1$ by using the slope computed in Section 3 . By substituting $log_2(\#text) = 108$, Algorithm 2 can break RC6W with 15 rounds with $2^{102.4}$ plaintexts with a probability of 90%. Algorithm 2 can work faster than an exhaustive key search with 2^{20} memory.

4.2 Algorithm 3

We improve the Algorithm 2 by making use of the results of Test2. The conditions on plaintexts in the Algorithm 3 is: both B_0 and D_0 introduce zero rotation in the 1st round; and both $lsb_5(A_0)$ and $lsb_5(C_0)$ are just fixed.

Table 9. Success probability and the χ^2-value of Algorithm 3 (in 100 keys)

#texts	#keys	χ^2-value(63 degree)		
		Average	Level	Variance
2^{22}	21	63.067	0.526	0.003
2^{23}	54	63.135	0.528	0.003
2^{24}	93	63.267	0.533	0.005

Table 10. $\log_2$(#texts) required for recovering a key in Algorithm 3 (in 100 keys)

	90%	70%	30%
$\log_2(\#text)$	23.9	23.3	22.5

Algorithm 3

```
Recover $lsb_2(S_{2r})$ and $lsb_2(S_{2r+1})$ of RC6W.
Set $(lsb_3(B_{r+1}), lsb_3(D_{r+1})) = (y_b, y_d)$, $(lsb_2(S_{2r}), lsb_2(S_{2r+1})) = (s_a, s_c)$,
and $(lsb_5(F(A_{r+1})), lsb_5(F(C_{r+1}))) = (x_c, x_a)$.
1. Choose a plaintext $(A_0, B_0, C_0, D_0)$
   with $(lsb_5(F(B_0)), lsb_5(F(D_0)), lsb_5(C_0)) =$
   $(0, 0, 0)$, set $lsb_5(A_0) = t$, and encrypt it.
2. For each $s_a, s_c = 0, 1, 2, 3$, set $s = s_a||s_c$, $S^3_{2r}, S^3_{2r+1} = 0$, and decrypt $y_d||y_b$
   with a key $(S^3_{2r}||s_a, S^3_{2r+1}||s_c)$ by 1 round, which are set to $z_a||z_c = z$.
3. For each value $s$, $t$, $x_a$, $x_c$, and $z$,
   increment each array $count[s][t][x_a][x_c][z]$.
4. For each $s$, $t$, $x_a$, $x_c$, compute $\chi^2[s][t][x_a][x_c]$.
5. Compute the average $ave[s]$ of $\{\chi^2[s][t][x_a][x_c]\}$ for each $s$, and output $s$
   with the highest $ave[s]$ as $lsb_2(S_{2r})||lsb_2(S_{2r+1})$.
```

The number of available plaintexts in Algorithm 3 is 2^{113}. Algorithm 3 uses plaintexts with $lsb_5(C_0) = 0$, but this condition is further eased by classifying the value of $lsb_5(C_0)$. Then the number of available plaintexts becomes 2^{118}.

Table 9 shows the results for RC6W with 5 rounds: the success probability among 100 trials, the average of χ^2-value of recovered keys, the level, and the variance. Let us compare the results with that of Algorithm 2 in Table 7. In Algorithm 3, the plaintexts computed on the χ^2-value is further classified to each group by the value of $lsb_5(A_0)$. Since all plaintexts in our experiments are randomly generated by m-sequences, plaintexts are roughly estimated to be uniformly distributed to each group. Therefore, the χ^2-test of using $2^{22} - 2^{24}$ plaintexts in Algorithm 3 corresponds to that of $2^{17} - 2^{19}$ in Algorithm 2. In the same way, the χ^2-test of using $2^{22} - 2^{24}$ plaintexts in Algorithm 3 corresponds to that of $2^7 - 2^9$ in the case of $n = 3$ of Test 3. We see the average of χ^2-value by using 2^{22}, 2^{23}, or 2^{24} in Table 9 is roughly equal to that by using 2^{17}, 2^{18}, or 2^{19} in Table 7, and that by using 2^7, 2^8, or 2^9 in Table 4, respectively. On the other hand, the variance of χ^2-value by using 2^{22}, 2^{23}, or 2^{24} in Table 9 is about $1/2^5$ as much as that by using 2^{17}, 2^{18}, or 2^{19} in Table 7, and about $1/2^{15}$ as much as that by using 2^7, 2^8, or 2^9 in Table 4, respectively. Algorithm 3 keeps the level of the average of χ^2-value with less variance of χ^2-value. As a result, Algorithm 3 can recover a key with more low level by reducing the variance of χ^2-value than Algorithm 2.

More precise experimental results are shown in Table 10. All experiments are calculated to the first decimal place. From Table 10, the number of plaintexts required for recovering a key in r rounds with the success probability of 90%, $log_2(\#text)$, is estimated to $log_2(\#text) = 8.1r-16.6$ by using the bias computed in Section 3 . By substituting $log_2(\#text) = 118$, Algorithm 2 can break RC6W with 16 rounds with $2^{113.0}$ plaintexts with a probability of 90%. Algorithm 3 can work faster than an exhaustive key search with 2^{25} memory.

4.3 Algorithm 4

We improve Algorithm 3 by using all plaintexts and classifying them into the same $lsb_3((A_0 \oplus F(B_0)) \lll F(D_0))$ and $lsb_3((C_0 \oplus F(D_0)) \lll F(B_0))$(see the results of Test 5 and 6 in Section 3).

Algorithm 4

```
Recover $lsb_2(S_{2r})$ and $lsb_2(S_{2r+1})$ of RC6W.
    Set $(lsb_3(B_{r+1}), lsb_3(D_{r+1})) = (y_b, y_d)$, $(lsb_2(S_{2r}), lsb_2(S_{2r+1})) = (s_a, s_c)$,
    and $(lsb_5(F(A_{r+1})), lsb_5(F(C_{r+1})) = (x_c, x_a)$.
1. Given a plaintext $(A_0, B_0, C_0, D_0)$, set $lsb_3((A_0 \oplus F(B_0)) \lll F(D_0)) = t_a$,
    $lsb_3((C_0 \oplus F(D_0)) \lll F(B_0)) = t_c$, and encrypt it.
2. For each $s_a, s_c = 0, 1, 2, 3$, set $s = s_a||s_c$, $S^3_{2r}, S^3_{2r+1} = 0$, and decrypt $y_d||y_b$
    with a key $(S^3_{2r}||s_a, S^3_{2r+1}||s_c)$ by 1 round, which are set to $z_a||z_c = z$.
3. For each value $s$, $t_a$, $t_c$, $x_a$, $x_c$, and $z$,
    increment $count[s][t_a][t_c][x_a][x_c][z]$.
4. For each $s$, $t_a$, $t_c$, $x_a$, $x_c$, compute $\chi^2[s][t_a][t_c][x_a][x_c]$.
5. Compute the average $ave[s]$ of $\{\chi^2[s][t_a][t_c][x_a][x_c]\}$ for each $s$,
    and output $s$ with the highest $ave[s]$ as $lsb_2(S_{2r})||lsb_2(S_{2r+1})$.
```

The number of available plaintexts in Algorithm 4 is 2^{128}. Algorithm 4 classifies plaintexts by each 3 bit of $(A_0 \oplus F(B_0)) \lll F(D_0)$ and $(C_0 \oplus F(D_0)) \lll F(B_0)$, which may be enlarged to, for example, 5, like the conditions of Test 5 and 6. However, the larger classified bit size is, the larger memory is required.

Table 11 show the results for RC6W with 5 rounds: the success probability among 100 trials, the average of χ^2-value of recovered keys, the level, and the variance. We see that, in Algorithm 4, the variance of χ^2-value is much more reduced than Algorithm 2 and 3. As a result, Algorithm 4 can recover a key more efficiently by reducing the variance of χ^2-value than Algorithm 2 and 3.

More precise experimental results are shown in Table 12. All experiments are calculated to the first decimal place. From Table 12, the number of plaintexts required for recovering a key in r rounds with the success probability of 90%, $log_2(\#text)$, is estimated to $log_2(\#text) = 8.1r - 13.8$, by using the slope computed in Section 3 . By substituting $log_2(\#text) = 128$, Algorithm 4 can break RC6W with 17 rounds by using $2^{123.9}$ plaintexts in a probability of 90%. Algorithm 4 can work faster than an exhaustive key search with 2^{26} memory.

5 A Key Recovery Algorithm against RC6-64

This section applies Algorithm 4 to a two-register version of RC6 without whitening, RC6-64W, and discusses the security difference between RC6-64 and RC6.

Table 11. Success probability and the χ^2-value of Algorithm 4 (in 100 keys)

#texts	#keys	χ^2-value(63 degree)		
		Average	Level	Variance
2^{25}	26	63.057	0.526	0.0003
2^{26}	59	63.108	0.528	0.0005
2^{27}	100	63.230	0.532	0.0007

Table 12. $\log_2(\#text)$ required for recovering a key in Algorithm 4(in 100 keys)

	90%	70%	30%
$\log_2(\#text)$	26.7	26.3	25.3

We apply Algorithm 4 to RC6-64W after presenting RC6-64. The round function of RC6-64 is almost the same structure with that of RC6. An input of the i-th round is denoted by (A_i, B_i), and (A_0, B_0) is a plaintext, where each A_i and B_i is 64 bits. The i-th subkey S_i is also 64 bits. Here the function F is modified to F_6 in a 64-bit-oriented manner, $F_6(X) = X(2X+1) \pmod{2^{64}} \lll 6$.

Algorithm 5 (Encryption with RC6-64)

```
1. [pre-whitening] $A_1 = A_0$; $B_1 = B_0 + S_0$;
2. for $i = 1$ to $r$: $t = F_6(B_i)$; $A_i = ((A_i \oplus t) \lll t) + S_i$; $A_{i+1} = B_i$;$B_{i+1} = A_i$;
3. [post-whitening] $A_{r+2} = A_{r+1} + S_{r+1}$; $B_{r+2} = B_{r+1}$.
```

Algorithm 6 (Algorithm to RC6-64W)

```
This algorithm recovers $lsb_4(S_r)$ of RC6-64W.
Set $lsb_5(B_{r+1}) = y$, $lsb_4(S_r) = s$, and $lsb_6(F_6(A_{r+1})) = x$.
1. Given a plaintext $(A_0, B_0)$, set $lsb_5((A_0 \oplus F_6(B_0)) \lll F_6(B_0)) = t$,
   and encrypt.
2. For each $s$ $(s = 0, \cdots, 15)$, set $S_r^5 = 0$, and decrypt $y$ with the key $S_r^5||s$
   by 1 round. We also set a decryption of $y$ to $z$,
   which is a 5-bit integer.
3. For each value $s$, $t$, $x$, and $z$, increment each array $count[s][t][x][z]$.
4. For each $s$, $t$, and $x$, compute $\chi^2[s][t][x]$.
5. Compute the average $ave[s]$ of $\{\chi^2[s][t][x]\}$ for each $s$, and
   output $s$ with the highest $ave[s]$ as $lsb_4(S_r)$.
```

The number of available plaintexts in Algorithm 6 is 2^{128}. Table 13 shows the results for RC6-64W with 5 and 7 rounds: the success probability among 100 trials, the average of χ^2-value of recovered keys, the level, and the variance. More precise experimental results are shown in Table 14. All experiments are calculated to the first decimal place. From Table 14, the number of plaintexts required for recovering a key in r rounds with the success probability of 90%, $log_2(\#text)$, is estimated to $log_2(\#text) = 5.0r - 8.2$. By substituting $log_2(\#text) = 128$, Algorithm 6 can break RC6-64W with 27 rounds with $2^{126.8}$ plaintexts with a probability of 90%. Algorithm 6 can work faster than an exhaustive key search with 2^{20} memory.

We discuss the difference between the round function of RC6 and that of RC6-64 from the security point of view. First we conduct the following Test 7 of RC6-64, whose results are shown in Table 15.

Table 13. Success probability and the χ^2-value of Algorithm 6 (in 100 keys)

5 rounds					7 rounds				
#texts	#keys	χ^2-value(63 degree)			#texts	#keys	χ^2-value(63 degree)		
		Average	Level	Variance			Average	Level	Variance
2^{15}	20	31.214	0.545	0.0296	2^{25}	30	31.278	0.548	0.0394
2^{16}	65	31.504	0.559	0.0290	2^{26}	53	31.512	0.559	0.0302
2^{17}	96	32.022	0.584	0.0335	2^{27}	95	32.050	0.586	0.0286

Table 14. $\log_2$(#texts) required for recovering a key in Algorithm 6 (in 100 keys)

	5 rounds			7 rounds		
	90%	70%	30%	90%	70%	30%
$\log_2(\#text)$	16.8	16.2	15.3	26.9	26.2	25.0

Table 15. The χ^2-value on $lsb_5(A_{r+1})$ in Test 7(in 100 keys)

4 rounds									
# texts	$2^{6.9}$			$2^{8.7}$			$2^{9.5}$		
The χ^2-value	Average	Level	Variance	Average	Level	Variance	Average	Level	Variance
	34.600	0.700	86.071	40.893	0.890	126.840	51.261	0.988	188.444
6 rounds									
# texts	$2^{16.5}$			$2^{17.5}$			$2^{18.9}$		
The χ^2-value	Average	Level	Variance	Average	Level	Variance	Average	Level	Variance
	33.966	0.674	73.204	37.666	0.809	112.739	45.193	0.952	131.131

Test 7: χ^2-test on $lsb_5(A_{r+1})$ in RC6-64 with r rounds in the case which B_0 introduces the zero rotation in the 1st round, and $lsb_5(A_0) = 0$

Let us compare each round function between RC6-64 and RC6 by using Table 15 and 2. The security level of one round in RC6-64, the size of one-round subkeys, is equal to that in RC6-32, which has two 32-bit subkeys in one round. The round function of RC6-64 is almost the same structure as that of RC6. However, the slope, defined in Section 3.4, of RC6-64 is apparently lower than that of RC6. This means that the correlations between an input of round function and the output in RC6-64 is kept more than that in RC6. The round function of RC6-64 mixes up data less than that of RC6. The weakpoint of RC5 is thought to be a data dependent rotation, which is defined by only 5 bits of subkey and data. Although the weakness of data dependent rotation is improved in both RC6 and RC6-64, RC6-64 is much weaker than RC6. The difference between RC6-64 and RC6 is the data structure: RC6-64 consists of 2 units, and RC6 consists of 4 units. Both RC6-64 and RC6 make use of modular-additions in order to mix within the unit. Apparently, correlations are introduced by the consecutiveness of modular-additions. From our results, we see that the structure of RC6, 4-unit plaintexts, reduce correlations more efficiently than that of RC6-64, 2-unit plaintexts.

6 Conclusions

We have proposed an efficient and feasible known plaintext correlation attack on RC6W. Our attack can break RC6W/r with a success probability of 90% by using $2^{8.1r-13.8}$ plaintexts, and thus can break RC6W with 17 rounds by using $2^{123.9}$ plaintexts. We have analyzed that the security of RC6 is further enhanced by dividing data into 4 units to break the consecutiveness of modular additions.

References

1. A. Biryukov, and E. Kushilevitz, "Improved Cryptanalysis of RC5", *Advances in Cryptology-Proceedings of EUROCRYPT'98*, Lecture Notes in Computer Science, **1403**(1998), Springer-Verlag, 85-99.
2. J. Borst, B. Preneel, and J. Vandewalle, "Linear Cryptanalysis of RC5 and RC6", *Proceedings of Fast Software Encryption*, Lecture Notes in Computer Science, **1636**(1999), Springer-Verlag, 16-30.
3. S. Contini, R. Rivest, M. Robshaw, and Y. Yin, "Improved analysis of some simplified variants of RC6", *Proceedings of Fast Software Encryption*, Lecture Notes in Computer Science, **1636**(1999), Springer-Verlag, 1-15.
4. J. Hayakawa, T. Shimoyama, and K. Takeuchi, "Correlation Attack to the Block Cipher RC5 and the Simplified Variants of RC6", submitted paper in Third AES Candidate Conference, April 2000.
5. B. Kaliski, and Y. Lin, "On Differential and Linear Cryptanalysis of the RC5 Encryption Algorithm", *Advances in Cryptology-Proceedings of CRYPTO'95*, Lecture Notes in Computer Science, **963**(1995), Springer-Verlag, 171-184.
6. J. Kelsey, B. Schneier, and D. Wagner, "Mod n Cryptanalysis, with applications against RC5P and M6", *Proceedings of Fast Software Encryption*, Lecture Notes in Computer Science, **1636**(1999), Springer-Verlag, 139-155.
7. L. Knudsen, and W. Meier, "Improved Differential Attacks on RC5", *Advances in Cryptology-Proceedings of CRYPTO'96*, Lecture Notes in Computer Science, **1109**(1996), Springer-Verlag, 216-228.
8. L. Knudsen, and W. Meier, "Correlations in RC6 with a reduced number of rounds", *Proceedings of Fast Software Encryption*, Lecture Notes in Computer Science, **1978**(2001), Springer-Verlag, 94-108.
9. D. Knuth, *The art of computer programming*, *vol. 2*, *Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, Mass. 1981.
10. A. Menezes, P. C. Oorschot and S. Vanstone, *Handbook of applied cryptography*, CRC Press, Inc., 1996.
11. A. Miyaji, M. Nonaka and Y. Takii, "Improved Correlation Attack on RC5," *IEICE Trans.*, Fundamentals., *vol. E85-A*, No.1(2002), 44-57.
12. http://cryptonessie.org
13. R. Rivest, M. Robshaw, R. Sidney and Y. Yin, "The RC6 Block Cipher. v1.1", 1998.
14. R. Rivest, "The RC5 Encryption Algorithm", *Proceedings of Fast Software Encryption*, Lecture Notes in Computer Science, **1008**(1995), Springer-Verlag, 86-96.
15. S. Shirohata, *An introduction of statistical analysis*, Kyouritu Syuppan, 1992, (in Japanese).
16. T. Shimoyama, M. Takenaka, and T. Koshiba, "Multiple linear cryptanalysis of a reduced round RC6", *Proceedings of Fast Software Encryption*, Lecture Notes in Computer Science, Springer-Verlag, to appear.

Author Index