Joaquin Garcia-Alfaro Frédéric Cuppens
Nora Cuppens-Boulahia Ali Miri
Nadia Tawbi (Eds.)

LNCS 7743

Foundations and Practice of Security

5th International Symposium, FPS 2012
Montreal, QC, Canada, October 2012
Revised Selected Papers

Lecture Notes in Computer Science 7743

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

Editorial Board

Joaquin Garcia-Alfaro Frédéric Cuppens
Nora Cuppens-Boulahia Ali Miri
Nadia Tawbi (Eds.)

Foundations and Practice of Security

5th International Symposium, FPS 2012
Montreal, QC, Canada, October 25-26, 2012
Revised Selected Papers

Springer

Volume Editors

Joaquin Garcia-Alfaro
TELECOM SudParis, 91011 Evry CEDEX, France
E-mail: joaquin.garcia-alfaro@acm.org

Frédéric Cuppens
Nora Cuppens-Boulahia
TELECOM Bretagne, 35512 Cesson Sévigné CEDEX, France
E-mail:{frederic.cuppens, nora.cuppens}@telecom-bretagne.eu

Ali Miri
Ryerson University, Toronto, ON, M5B 2K3, Canada
E-mail: ali.miri@ryerson.ca

Nadia Tawbi
Université Laval, Quebec, QC, G1V 0A6, Canada
E-mail: nadia.tawbi@ift.ulaval.ca

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-37118-9 e-ISBN 978-3-642-37119-6
DOI 10.1007/978-3-642-37119-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013934231

CR Subject Classification (1998): E.3, E.4, K.6.5, C.2.0, D.4.6

LNCS Sublibrary: SL 4 – Security and Cryptology

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

Printed on acid-free paper

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

Preface

This volume contains the revised version of the papers presented at the 5th International Symposium on Foundations and Practice of Security (FPS 2012). The symposium was held at ETS (École de technologie supérieure), in Montréal, Canada, during October 25–26, 2012. The event covered a wide and rich spectrum of research in different areas of theoretical and practical security solutions for information systems.

In response to the call for participation, 62 papers from 28 different countries were submitted to FPS 2012. These submission were evaluated on the basis of their significance, novelty, and technical quality. All submissions went through a careful anonymous review process (three or more reviews per submission) aided by 55 Technical Program Committee members and 32 external referees. In the end, 22 full papers, accompanied by three short papers, were presented at the event. The final program also included two invited talks by Douglas Stinson (University of Waterloo, Canada) and Ana Rosa Cavalli (TELECOM & Management SudParis, France). Our special thanks to Douglas and Ana for accepting our invitation and for their presence during the event and talks.

We would like to thank everyone who gave his or her time, energy, and ideas to assist in organizing this event, including all the members of the Organizing Committee, Program Committee members, and all the external referees. In particular, we would like to highlight and acknowledge the tremendous efforts of Chamseddine Talhi (Local Arrangements Chair), who worked tirelessly on various symposium-related tasks. Our gratitude goes also to Evangelos Kranakis and Michel Barbeau, for their unconditional help since the beginning of this event. We also thank art designer Berta Mir Daza, for all her help, availability, and commitment. Thanks very much for your contribution to the success of the event.

Many thanks go to the sponsors who made this event possible: the Fields Institute for Research in Mathematical Sciences, ETS (École de technologie supérieure), Ryerson University, Institut Mines-Télécom, Labsticc (Laboratoire en sciences et techniques de l'information, de la communication et de la connaissance), the IN3 research institute of the Open University of Catalonia, and the Montréal section of the IEEE. We also acknowledge the support of the following projects of the Spanish MICINN: TSI2007-65406-C03-03 E-AEGIS, TIN2011-27076-C03-02 CO-PRIVACY and CONSOLIDER INGENIO 2010 CSD2007-0004 ARES.

Last but by no means least we thank to all the authors who submitted papers, and to all the symposium attendees.

November 2012 FPS 2012 Chairs

Organization

General Chairs

Frédéric Cuppens	TELECOM Bretagne, France
Ali Miri	Ryerson University, Canada

Program Chairs

Nora Cuppens-Boulahia	TELECOM Bretagne, France
Joaquin Garcia-Alfaro	TELECOM SudParis, France
Nadia Tawbi	Université Laval, Canada

Local Arrangements Chair

Chamseddine Talhi	École de technologie supérieure, Canada

Organizing Chairs

Esma Aimeur	Université de Montréal, Canada
Chamseddine Talhi	École de technologie supérieure, Canada
Nadia Tawbi	Université Laval, Canada

Program Committee

Gildas Avoine	Catholic University of Louvain, Belgium
Diala Abihaidar	Dar Al Hekma College, Saudi Arabia
Carlisle Adams	Ottawa University, Canada
Kamel Adi	Université du Québec en Outaouais, Canada
Esma Aimeur	Université de Montréal, Canada
Michel Barbeau	Carleton University, Canada
Hanifa Boucheneb	Polytechnique Montréal, Canada
Jordi Castella-Roca	Rovira i Virgili University, Spain
Ana Cavalli	TELECOM SudParis, France
Frédéric Cuppens	TELECOM Bretagne, France
Nora Cuppens-Boulahia	TELECOM Bretagne, France
Vanesa Daza	Universitat Pompeu Fabra, Spain
Mourad Debbabi	Concordia University, Canada
Roberto Di Pietro	Roma Tre University of Rome, Italy

Nicola Dragoni	Technical University of Denmark, Denmark
David Evans	University of Derby, UK
Marc Frappier	Université de Sherbrooke, Canada
José M. Fernández	École Polytechnique de Montréal, Canada
Sara Foresti	University of Milan, Italy
Martin Gagné	Joseph Fourier University, France
Sebastien Gambs	University of Rennes 1, France
Flavio D. Garcia	Radboud University Nijmegen, The Netherlands
Joaquin Garcia-Alfaro	TELECOM SudParis, France
Abdelwahab Hamou-Lhadj	Concordia University, Canada
Jordi Herrera-Joancomarti	Autonomous University of Barcelona, Spain
Bruce Kapron	University of Victoria, Canada
Evangelos Kranakis	Carleton University, Canada
Hyoungshick Kim	University of British Columbia, Canada
Pascal Lafourcade	Joseph Fourier University, France
Yassine Lakhnech	Joseph Fourier University, France
Georgios Lioudakis	National Technical University of Athens, Greece
Giovanni Livraga	University of Milan, Italy
Luigi Logrippo	Université du Québec en Outaouais, Canada
Javier Lopez	University of Malaga, Spain
Joan Melia-Segui	Universitat Pompeu Fabra, Spain
Mohamed Mejri	Université Laval, Canada
Ali Miri	Ryerson University, Canada
Guillermo Navarro-Arribas	Autonomous University of Barcelona, Spain
Jordi Nin	Universitat Politecnica de Catalunya, Spain
Melek Onen	Eurecom, France
Andreas Pashalidisi	K. U. Leuven, Belgium
Silvio Ranise	FBK, Security and Trust Unit, Italy
Jean-Marc Robert	École de technologie supérieure, Canada
Rei Safavi-Naini	Calgary University, Canada
Claudio Soriente	ETH Zurich, Switzerland
Alessandro Sorniottia	IBM Research Zurich, Switzerland
Anna Squicciarini	Penn State University, USA
Douglas Stinson	University of Waterloo, Canada
Chamseddine Talhi	École de technologie supérieure, Canada
Issa Traore	University of Victoria, Canada
Carmela Troncoso	K.U. Leuven, Belgium
Nadia Tawbi	Université Laval, Canada
Alexandre Viejo	Rovira i Virgili University, Spain
Lena Wiese	University of Hildesheim, Germany
Nicola Zannone	Eindhoven University of Technology, The Netherlands

External Referees

Khalifa Toumi	TELECOM SudParis, France
Stere Preda	Concordia University, Canada
Cédric Lauradoux	INRIA Rhone-Alpes, France
Arnau Vives-Guasch	Rovira i Virgili University, Spain
Cristina Romero-Tris	Rovira i Virgili University, Spain
Marcelo Brocardo	University of Victoria, Canada
Abdelfattah Amamra	École de technologie supérieure, Canada
Samir Ouchani	Concordia University, Canada
Jan Stanek	IBM Research Zurich, Switzerland
Roel Verdult	Radboud University Nijmegen, The Netherlands
Bassam Sayed	University of Victoria, Canada
Maria Koukovini	National Technical University of Athens, Greece
Josep Balasch	K.U. Leuven, Belgium
Cesar Andres Sanchez	Universidad Complutense de Madrid, Spain
Anis Bkakria	TELECOM Bretagne, France
Stere Preda	Concordia University, Canada
Sherif Saad	University of Victoria, Canada
Robert Warren	Carleton University, Canada
Elias Bou-Harb	Concordia University, Canada
Kris Haralambiev	IBM Research Zurich, Switzerland
Mohsen Alimomeni	University of Calgary, Canada
Alessio Di Mauro	Technical University of Denmark, Denmark
Behzad Malek	Ottawa University, Canada
Mahavir Jhawar	University of Calgary, Canada
Elisa Costante	Eindhoven University of Technology, The Netherlands
Eugenia Papagiannakopoulou	National Technical University of Athens, Greece
Mina Askari	University of Calgary, Canada
Alfredo Rial	K. U. Leuven, Belgium
Ai Ho Thanh	Université de Montréal, Canada
Felipe Lalanne	TELECOM SudParis, France
Davide Papini	Technical University of Denmark, Denmark
Tania Martin	Catholic University of Louvain, Belgium

Table of Contents

Cryptography and Information Theory

Key Management and Cryptographic Protocols

Privacy and Trust

Policies and Applications Security

Network and Adaptive Security

Short Papers

MaD2: An Ultra-Performance Stream Cipher for Pervasive Data Encryption

Jie Li[1] and Jianliang Zheng[2]

[1] Department of Computer Science, Graduate Center,
City University of New York, New York, NY, USA
[2] Independent Scholar, New York, NY, USA
zheng@ee.ccny.cuny.edu

Abstract. MaD2 is an ultra-performance stream cipher that runs into one clock cycle per byte on a typical personal computer. With an encryption/decryption rate significantly higher than the disk data transfer rate, it can be employed to secure data at rest with almost no user observable performance degradation. The cipher resists various known cryptanalytic attacks. Its keystream demonstrates good statistical properties and clears all the NIST statistical tests, the new Diehard battery of tests, and the TestU01 batteries of tests.

Keywords: Stream Cipher, Data Encryption, Encryption at Rest, High Performance, Cloud Computing.

1 Introduction

Pervasive data encryption means two things: encrypt all data and encrypt data while they are moving on the wire and at rest as well. Not all data are secrets and need to be secured, but such a need arises when data classification is difficult or expensive (e.g., in cloud computing) or when encryption needs to be done at a layer below the application (e.g., for whole disk or communication channel encryption). Encryption at rest refers to the fact that the data is physically stored in an encrypted format. This is different from encryption in flight, which is only applied to data to be transported. Encryption at rest may affect the usability of some applications, but it is getting more popular nowadays for several reasons. First, most computers and computing devices are connected to networks and face attacks that keep growing in both quantity and complexity. Second, the boom of virtual computing and cloud computing makes it difficult or impossible to put up a physical defense line against attacks. Third, the risk of losing sensitive data increases due to the wide use of portable computing devices, which are more likely to get lost or be stolen. Finally, the breach of data at rest usually has a more serious consequence than the breach of data on the wire, since the former can contain much more information than the latter.

Past several years have witnessed significant advances in data encryption, including the release of Advanced Encryption Standard (AES) [1] and the development of eStream project (http://www.ecrypt.eu.org/stream/). Nonetheless,

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 1–17, 2013.

data encryption is still a costly operation and its performance penalty is still too high. For example, the data encryption speed of AES on an Intel Core i3 personal computer is about two times slower than the speed of disk I/O. This means data I/O performance will be degraded by a factor of 3 when data are encrypted using AES. Stream ciphers such as RC4 [2] and eStream candidates [3,4,5,6] are faster than block cipher AES, but they still slow down data access by a factor of 1.5 or more. The fast development of the Internet and the ongoing transition from private computing to cloud computing call for more data encryption and more efficient data encryption. In light of this, we developed a new stream cipher called MaD2, which is several times faster than any existing ciphers we know. As we will show, this new cipher resists various known cryptanalytic attacks. It also demonstrates good statistical feature and passes all the NIST statistical tests [7], the new Diehard battery of tests [8] including the "tough" tests in [9], and the TestU01 batteries of tests [10].

The rest of this paper is structured as follows. In section 2, we describe the algorithm in detail. Next, in section 3, we present the security analysis of our stream cipher. Then, in section 4 and 5, we give the statistical testing results and performance testing results respectively. Finally, in section 6, we conclude with a summary.

2 Algorithm Details

In this section we present the algorithm details. We first give a brief review of RC4 [2], which will be modified and used as one of the building blocks in MaD2. We then cover the key scheduling, initialization of internal state, and keystream generation of MaD2. All components will be described using pseudo code and the following notations are used:

Notation	Usage
`#`	starting a comment line
`++`	increment (x`++` is same as $x = x + 1$)
`%`	modulo
`<<`	left logical bitwise shift
`>>`	right logical bitwise shift
`&`	bitwise AND
`|`	bitwise OR
`^`	bitwise XOR
`[]`	array subscripting (subscript starts from 0)

Hexadecimal numbers are prefixed by "0x" and all variables and constants are unsigned integers in little endian.

2.1 RC4

RC4 is simple and ideal for software implementation. It maintains an internal state, which consists of a permutation of all $2^8 = 256$ possible octets and two 8-bit indices for accessing elements in the permutation. The permutation

Listing 1. RC4

```
1  ## addition (+) and increment (++) operations ##
2  ## are performed modulo 256                    ##
3
4  # Key Scheduling Algorithm (KSA)
5  for i from 0 to 255
6      S[i] = i
7  endfor
8  j = 0
9  for i from 0 to 255
10     j = j + S[i] + key[i % keylength]
11     swap(S[i], S[j])
12 endfor
13
14 # Pseudo-Random Generation Algorithm (PRGA)
15 i = 0
16 j = 0
17 while GeneratingOutput
18     i++
19     j = j + S[i]
20     swap(S[i], S[j])
21     n = S[i] + S[j]
22     output S[n]
23 endwhile
```

is first initialized to the identity permutation and then shuffled using a secret key according to the key scheduling algorithm (KSA) given on lines 5 to 12 of Listing 1. Afterwards, pseudo-random number sequence is generated using the pseudo-random generation algorithm (PRGA) given on lines 15 to 23 of Listing 1. The PRGA iterates as many times as needed and during each iteration it continues to shuffle the permutation by swapping two elements and outputs one octet. To encrypt or decrypt a message, simply act the pseudo-random number sequence on the message using bitwise XOR.

2.2 Key Scheduling

Our key scheduling algorithm is modified from RC4 KSA and shown in Listing 2. It iterates 320 times instead 256 times to shuffle the identity permutation. In RC4 the first *keylength* bytes of S are likely to have similar patterns after key scheduling if similar keys that only differ at the end are used. This problem can be fixed by adding *keylength* iterations in additional to the original 256 iterations. But it is unwise to make the number of iterations depend on the key size, because it leaks information about the key size and can be exploited by a side-channel attack. Since the maximum allowed key size in MaD2 is 64 bytes (i.e., $keylength \leq 64$), we choose to add 64 iterations regardless of the actual key

Listing 2. Key Scheduling Algorithm

```
# addition (+) and increment (++) operations
# are performed modulo 256
for i from 0 to 255
    S[i] = i
endfor
i = 0
j = 0
k = 0
for r from 0 to 319
    j = j + S[i] + key[i % keylength]
    k = k ^ j
    left_rotate(S[i], S[j], S[k])
    i++
endfor
i = j + k
```

size. Note that we use a new 16-bit unsigned integer r instead of the 8-bit index i as the loop counter in the second loop. This is because we need to iterate 320 times but index i can only have 256 different values. Except r, all other variables are 8-bit unsigned integers. MaD2 introduces a third index k, which is initialized to 0 and then XORed with index j during each iteration. It also replaces the swap operation with a left rotation operation among $S[i]$, $S[j]$, and $S[k]$ (i.e., $tmp = S[i]$, $S[i] = S[j]$, $S[j] = S[k]$, $S[k] = tmp$), which in effect performs more swaps. At the end we set the value of index i to the sum of index j and k and persist all three indices for future use.

RC4 can accept a key that is as large as 256 bytes, but in practice a much smaller one is used. As aforementioned, the maximum key size in MaD2 is 64 bytes (512 bits). Out of the 64 bytes, we only claim security for 56 bytes. Like RC4, MaD2 does not specify the use of an initialization vector (IV) or a nonce, but an application can use an IV as part of the key in the key scheduling.

2.3 Initialization of Internal State

We maintain a data structure shown in Fig. 1, which comprises four 512-byte state tables, denoted as *Sa*, *Sb*, *Sc*, and *Sd*, and four 64-bit integers, denoted as *a*, *b*, *c*, and *d*. The first two state tables *Sa* and *Sb* and the four integers *a*, *b*, *c*, and *d* construct the internal state of our cipher. The other two state tables *Sc* and *Sd* are used for buffering keystream sequence generated from the internal state. The first 256 bytes of *Sa* are also referred to as state table S to indicate that they play the role of the permutation table S in key scheduling. The concatenation of *Sa* and *Sb* is sometimes used as a large 1024-byte state table, referred to as *Sw*.

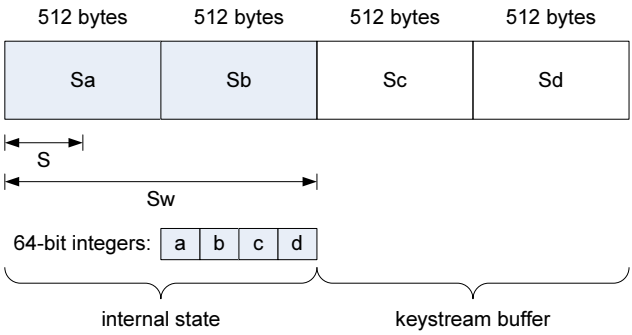


Fig. 1. Data Structure

To initialize the internal state, state table S (i.e., the first 256 bytes of Sa) is initialized according to the KSA given in Listing 2. Once initialized, S is further used to bootstrap the remaining part of the internal state. First the data in S are copied to the second half of Sa, then S is shuffled using the following code:

```
# addition (+) and increment (++) operations
# are performed modulo 256
for r from 0 to 255
    i++
    j = j + S[i]
    k = k ^ j
    left_rotate(S[i], S[j], S[k])
endfor
```

Note that we cannot use index i to control the above loop because we do not want to reset its value. The *copy-and-shuffle* procedure repeats until state table Sb is also populated. Finally 32 pseudo-random bytes are generated as follows:

```
# addition (+) and increment (++) operations
# are performed modulo 256
for r from 0 to 7
    i++
    j = j + S[i]
    k = k ^ j
    swap(S[i], S[j])
    m = S[j] + S[k]
    n = S[i] + S[j]
    output S[m]
    output S[n]
    output S[m ^ j]
    output S[n ^ k]
endfor
```

These 32 bytes are converted into four 64-bit integers using little endian and assigned to integer a, b, c, and d. This completes the initialization and the internal

state now contains four permutations of {0, 1, ..., 255} and four initialized 64-bit integers.

2.4 Keystream Generation

MaD2 takes advantages of modern 64-bit platforms and uses 64-bit operations to generate keystream. All state tables (*Sa*, *Sb*, *Sc*, *Sd*, and *Sw*) are cast into and used as 64-bit integer arrays in the keystream generation. When an encryption or decryption request is received, the keystream buffer (i.e., *Sc* + *Sd*) is checked. If it is not empty, the data stored in it are used to perform the required encryption or decryption operation. After all the data in the buffer are consumed, the cipher refreshes the buffer according to the keystream generation algorithm shown in Listing 3. Variable x is a byte array of size 64 used for indirection operation, that is, its bytes are used as indices to access state tables. It is computed from a, b, c, d, and two constants m and n. Each byte of x has a value falling in the range [0, 127] and any two bytes with a distance less than 4 have distinct values. *e, f, g,* and h are intermediate values computed at the beginning of and used throughout each keystream buffer refreshing operation (simply referred to as keystream refreshing henceforth). During each keystream refreshing, the algorithm iterates 64 times and two 64-bit integers are generated during each iteration, with one saved in *Sc* and the other in *Sd*. Note that, for each full block of data (1024 bytes), buffer refreshing can be skipped since encryption or decryption can be performed directly on the data.

The combination of $Sw[x[i]]$, $Sw[x[i]\hat{}0x7c]$, $Sa[ii]$, and $Sb[ii]$ introduces pseudo-random indirect access and guarantees all state table integers get involved during each keystream refreshing. The way we choose these four state table integers during each iteration deserves some explanations. First note that both $Sw[x[i]]$ and $Sw[x[i]\hat{}0x7c]$ can access either state table *Sa* or *Sb* but they can never access the same state table, which also means each state table has the same chance to be accessed by them. Integer ii is computed in such a way that its lower two bits cycle through the values 0, 1, 3, 2 instead of the normal 0, 1, 2, 3, but otherwise it is same as the counter i. The sequence 0, 1, 3, 2, when expressed in binary format 00, 01, 11, 10, has a characteristic that any two adjacent numbers differ by one single bit only. This arrangement, together with the use of constants m and n, results in a special feature, namely, these four state table integers are distinct and they are also different from any of the four state table integers used in the previous or next iteration. By distinct and different, we mean they point to different state table integers, which do not necessarily but with a high probability have different values. One can verify this feature by observing the following facts: $Sw[x[i]]$ and $Sw[x[i]\hat{}0x7c]$ are distinct; so are $Sa[ii]$ and $Sb[ii]$; the lower two bits of $x[i]$ and $x[i]\hat{}0x7c$ come from n and cycle through the values 3, 2, 0, 1, while those of ii cycle through the values 0, 1, 3, 2.

For efficiency and simplicity, only a few types of operations are used. They are bitwise AND, bitwise OR, bitwise XOR, addition, left logical bitwise shift, and right logical bitwise shift, each taking only one clock cycle for most processors

Listing 3. Keystream Generation

```
## additions are performed modulo ##
## 0x10000000000000000             ##

# declare a byte array of size 64
byte x[64]

# cast the byte array into 64-bit integer array
x[64] => x64[8]

# populate array x (through x64)
m = 0x7c7c7c7c7c7c7c7c
n = 0x0302000103020001
x64[0] = (a & m) | n
x64[1] = (b & m) | n
x64[2] = (c & m) | n
x64[3] = (d & m) | n
x64[4] = ((a >> 1) & m) | n
x64[5] = ((b >> 1) & m) | n
x64[6] = ((c >> 1) & m) | n
x64[7] = ((d >> 1) & m) | n

# compute e, f, g, and h
e = a + Sw[a >> 57]
f = b + Sw[b >> 57]
g = c + Sw[c >> 57]
h = d + Sw[d >> 57]

# output and update the internal state
ii = 0
for i from 0 to 63
    a = a << 1
    b = b >> 1
    ii = ii ^ i
    a = a + (e ^ Sw[x[i]])
    b = b + (f ^ Sw[x[i]^0x7c])
    c = c + (g ^ Sa[ii])
    d = d + (h ^ Sb[ii])
    ii = ii & 1
    Sc[i] = c ^ (a + d)
    Sd[i] = d ^ (b + c)
    Sw[x[i]] = a + b
endfor
```

when operands are immediate constants or register variables [11]. All four integers a, b, c, and d are updated during each iteration.

Besides integer a, b, c, and d, one integer from state table Sa or Sb is also updated via $Sw[x[i]]$ during each iteration. On average each integer has a 50% chance to get updated during each keystream refreshing. In other words, nearly half of Sa and Sb is updated during each keystream refreshing. Is it fast enough to update half of Sa and Sb during each keystream refreshing? The answer is "yes". Due to the shift operations, the value of a is determined by the most recent 64 values of $Sw[x[i]]$ (and the fixed e) used to update it and the value of b is largely determined by the most recent 64 values of $Sw[x[i]\hat{}0x7c]$ (and the fixed f) used to update it. On the other hand, c and d are permanently affected by any state table integer that has been involved in the computation of their values. This means the update of a single state table integer can completely change the evolution path of c and d. From the way how a, b, c, and d are computed, it suffices to only update some of the integers in Sa and Sb during one keystream refreshing.

3 Security Analysis

In this section we analyze the security of our cipher.

3.1 Period Length

For an n-bit internal state, the maximum possible period length is 2^n. Depending on the state transition algorithm, the actual period length may be much shorter. For pseudo-random mappings with no restrictions (simply referred to as pseudo-random mappings hereafter), the average period is about $2^{n/2}$. Given that MaD2 has a 8448-bit internal state and its state transition follows pseudo-random mappings, the expected period of the internal state is about $2^{4224} \approx 3.55 \times 10^{1271}$.

3.2 Resistance against Known Attacks

In this subsection, we give a cryptanalysis of our cipher in the context of known attacks, including special attacks mounted against RC4 and other generic attacks.

Attacks against RC4. RC4 key scheduling algorithm and keystream generation algorithm are both based on simple pseudo-random permutations. This approach makes RC4 one of the most efficient and popular stream ciphers so far. Nonetheless the simplicity of the cipher also leads to some security issues.

In [12], Roos described a class of weak keys in RC4. In [13], Golić derived a linear model of RC4 using the linear sequential circuit approximation method. According to this model, it requires about $64^n/225$ keystream words to detect the linear statistical weakness of RC4. This is significantly smaller than 2^M,where $M = n2^n + 2n$ is the bit size of RC4 internal state. In 2001, Fluhrer et al. described two significant weaknesses of RC4 key scheduling algorithm [14].

The first one is the existence of large classes of weak keys, whose length is divisible by some non-trivial power of two, i.e., $\ell = 2^q m$ for some $q > 0$. The second weakness is a related key vulnerability. The authors observed that when the same secret part of the key is used with numerous different exposed values, it takes relatively little work to rederive the secret part by analyzing the initial word of the keystreams. The significance of this finding is that many applications, including the Wired Equivalent Privacy (WEP) protocol, construct RC4 keys by concatenating a long term secret key with a varying but publically known IV, thus vulnerable to this related key attack. A strong correlation between the observable i, $S[n]$ and the internal j, $S[i]$, $S[j]$ was reported by Klein in [15]. This strong correlation improves the attack described in [14] and enabled Tews et al. to break 104-bit WEP in less than one minute [16].

There are other attacks reported in the open literature. The success of those attacks has revealed several design problems of RC4:

1. The key scheduling algorithm is too simple – 256 swaps are not enough to break the correlation between the input key and the initialized internal state.
2. The internal state evolves relatively simply and slowly. This helps transfer the initial correlation into later states and the keystream.
3. Permutations are reversible and easier to analyze than unrestricted irreversible mappings.

MaD2 uses rotation operations instead of swap operations to initialize the first 256 bytes of its internal state. This is more efficient and has a better mixing effect. In addition, limiting the maximum key size of MaD2 to 64 bytes helps prevent related key attacks, since any differences between two input keys will come into play within the first 64 iterations of key scheduling, giving more time for diffusion compared with a longer key that might be used with RC4. Diffusion is most effective when different keys result in different index j and/or k, which has a high probability to happen in MaD2. More importantly, the first 256 bytes of the internal state, once initialized, is further used to initialize the entire internal state to make sure sufficient diffusion is achieved. This overall design makes it very difficult to find simple correlation relationships between the input key and the initialized internal state as in [12,14,15]; nor is it possible to derive a linear model to approximate MaD2 as done for RC4 in [13].

MaD2 uses a more sophisticated keystream generation algorithm than RC4. Each output integer is computed from three integers (a, c, and d, or b, c, and d), all in turn computed from many state table integers. This can effectively prevent the correlation, if any, from being transferred into the keystream. Using pseudo-random mappings instead of pseudo-random permutations makes MaD2 more resistant to algebraic cryptanalysis. Since pseudo-random mappings are irreversible, MaD2 also possesses another feature that RC4 lacks, that is, knowing a state does not enable one to go back to its previous state. Knowing the keystream additionally does not help much either, since each output integer is computed from many internal integers and the computation involves indirect access (via $Sw[x[i]]$ and $Sw[x[i]\hat{}0x7c]$), which is nonlinear.

Time-Memory Tradeoff Attacks. Time-Memory tradeoff attacks rely on precomputation to reduce the effort needed for recovering the internal state and/or secret key [17]. This type of attacks proceed as follows: assume that the cipher is in a certain state and calculate a number of output bits and put the pair (output, state) in a sorted list; after enough pairs are calculated and stored, try to match a received output sequence with the saved output sequences; if the match is successful, then with some likelihood the internal state or partial of it may be determined, which may further lead to the recovery of the secret key. The parameters in a Time-Memory tradeoff attack are time (T), memory (M), and amount of output data (D). In general $T \times M^2 \times D^2 = S^2$, where S is the state space of the cipher and $D^2 \leq T$ [17]. The precomputation time P is computed as $P = S/D$. The design strength of MaD2 is 448 bits. For the brute-force equivalent attack with $T = 2^{448}$ and $D \leq \sqrt{T} = 2^{224}$, $M = S/D/\sqrt{T} \geq 2^{8448}/2^{224}/2^{224} = 2^{8000}$, that is, the lower bound on memory for the attack is 2^{8000} bits, which is simply impractical.

Guessing Attacks. Guessing attacks can be more efficient than brute-force search if a stream cipher is not designed properly [18]. The strategy for this type of attacks is to guess a small part of the internal state and then deduce the remaining part. This is particularly powerful when applied to word-based stream ciphers. The reason is that word-based stream ciphers have a relatively small number of internal words and any word guessed has a good chance to participate in the computation of next iteration if the algorithm is not designed with caution. The consequence is that more and more words get revealed and the cipher is eventually broken. MaD2 is designed to be resistant to this type of attacks.

To be successful, an attacker must be able to do two things, namely, be able to efficiently verify his guessing (guess and verify) and be able to determine more unknowns based on his guessing (guess and determine). While it may not come for free, it is usually assumed that an attacker has access to some of the keystream when launching a guessing attack. In MaD2, if an attacker knows some of the keystream, for example, the value of $Sc[i]$ at a certain moment, he can guess two of the three integers (a, c, and d) and then compute the third integer. If he also knows $Sd[i]$, he can further compute the value of b. The attacker needs to guess 128 bits to figure out the values of all four integers a, b, c, and d. Once the attacker knows a and b, he can compute $Sw[x[i]] = a + b$. To know $x[i]$ and therefore identify which integer is to be updated, he needs to guess another 5 bits (out of the 7 bits, the lower 2 bits are known apriori). So the attacker needs to guess 133 bits (128 bits if he chooses not to know $x[i]$) in total during the first iteration of guessing.

During the second iteration, the attacker needs to guess 128 bits like in the first iteration to figure out the new values of a, b, c, and d, and then another 128 bits to figure out the values of two of e, f, g, and h (and also two of the four state table integers $Sw[x[i]]$, $Sw[x[i]\hat{}0\text{x}7\text{c}]$, $Sa[ii]$, and $Sb[ii]$). Note that since the attacker only needs to find out two values (one must be c or d) so as to know all the values of a, b, c, and d during each iteration, it is not necessary for him to find

out all the values of e, f, g, and h. Also notice that the second 128 bits guessing is based on the fact that the four state table integers $Sw[x[i]]$, $Sw[x[i]\hat{}0x7c]$, $Sa[ii]$, and $Sb[ii]$ are distinct and they are also different from any of the four state table integers used in the previous iteration. If two integers, for example $Sw[x[i]]$ and $Sa[ii]$, are identical, then the attacker only needs to guess 64 bits instead of 128 bits. If a state table integer used in the previous iteration, for example the already known $Sw[x[i]]$, can appear in the next iteration, then the 128 bits guessing is also reduced to 64 bits. Here we have ignored the relatively small cost that is needed to make two integers point to the same state table integer (a probability of $\frac{1}{128} = 2^{-7}$ or a cost of 7 bits) or make a state table integer used in the previous iteration appear in the next iteration (a probability of $2 \times \frac{1}{128} = 2^{-6}$ or a cost of 6 bits; the coefficient 2 comes from that each of the two state table integers whose values need to be determined can take the known value).

During the third iteration, the attacker still needs to find out the new values of two of the four integers a, b, c, and d. To achieve this, he needs to know the values of two of the four state table integers $Sw[x[i]]$, $Sw[x[i]\hat{}0x7c]$, $Sa[ii]$, and $Sb[ii]$. He does not need to guess 128 bits, however, because he already knows the values of three distinct state table integers, one during the first iteration and two during the second iteration. The one whose value is found during the first iteration has a probability of 2^{-5} to appear in the third iteration, thus reducing the workload from 128 bits to $64 + 5 = 69$ bits.

The first three iterations require more than 448 bits of work, which is our design strength. But let us go a little further to see what the cost the attacker needs to pay if he continues. During the fourth iteration, all the three state table integers whose values are found during the first two iterations have a chance to reappear. But since one has already reappeared in the third iteration (otherwise the third iteration requires a 128 instead of 69 bits guessing), only the two state table integers whose values are found during the second iteration can reappear in the fourth iteration[1]. The probability that they both reappear is $\frac{1}{32} \times \frac{1}{32} = 2^{-10}$. This shows that the attack cost is only 10 bits during the fourth iteration.

The above attack is not unique and different attack strategies can be taken, but none is likely to be more efficient than the above one. To conclude this analysis, we also want to point out that it is infeasible to break the 64-bit integers into smaller units so as to reduce the attack cost. If the smaller units, say bytes, can be computed independently, then the attack cost will be significantly reduced. This is because each 64 bits can be reduced to, for example, eight 8 bits, which is equivalent to 11 bits only ($8 \times 2^8 = 2^{11}$).

Algebraic Attacks. So far algebraic attacks on stream ciphers are mainly applied to those whose internal state is updated in a linear way. Most LFSR-based stream ciphers fall into this category. A typical LFSR-based stream cipher consists of an internal state S, a linear state update function L, and a nonlinear

[1] For this to happen, the attacker must not have chosen to work on $Sa[ii]$ and $Sb[ii]$, since the value of ii cannot repeat during one keystream refreshing.

output function f. Let S_0 denote the initial internal state at time $t = 0$, then at time t the internal state is $L^t(S_0)$ and the keystream output is $z_t = f(L^t(S_0))$. The goal of an algebraic attack is to recover the initial state or the secret key by coming up with and solving a system of nonlinear equations based on the algebraic relations between the internal state bits (or secret key bits) and the observed keystream bits.

MaD2 is word-based and both its state update function and output function are nonlinear. The analysis of this type of stream ciphers against algebraic attacks is more difficult than those for LFSR-based stream ciphers. The only related work we can find is by Wong et al., who analyzed the RC4 family of stream ciphers against algebraic attacks in [19]. The analysis and experiment results from the authors suggest that RC4 is most likely immune to algebraic attacks at present.

MaD2 shares some common features with RC4: The operations of MaD2 include word addition; its key scheduling algorithm is based on permutation; its keystream generation involves indirect access. Aside from those common features, MaD2 is different from RC4. First, MaD2 has a huge internal state that is substantially larger than that of RC4, meaning during an algebraic attack more equations (and likely of higher degrees) need to be handled and more keystream bits need to be collected. Second, MaD2 switches from pseudo-random permutations to pseudo-random mappings once the state initialization is complete. The simple algebraic structure of pseudo-random permutations has helped the authors to construct the equations and reduce their degrees in [19]. Building and solving equations for MaD2 will be more difficult and costly. Finally, the keystream generation of MaD2 is more complex and involves more operations than that of RC4, which will also make algebraic attacks more difficult for MaD2 than for RC4.

Distinguishing Attacks. A distinguishing attack technique that targets stream ciphers using linear masking was proposed by D. Coppersmith et al. and applied to SNOW 1.0 [20]. A stream cipher usually includes some nonlinear process in its design. The nonlinear process resembles a block cipher and its states are deemed uncorrelated if they are far away in time. Linear masking tries to mask the correlation among states close in time. It masks those states using independent parts of a linear process. The basic idea of the attack is to find some linear combination of the linear process that vanishes. When this same combination is applied to the output stream, the linear process would vanish. This way the attacker is left with the nonlinear process only, for which he can further look for a characteristic that can be distinguished from randomness. Distinguishing attacks have also been mounted for other stream ciphers that use linear masking, including SNOW 2.0 and Sosemanuk [21,22,23]. MaD2 does not use linear masking, thus rendering this type of distinguishing attacks irrelevant.

Due to the complex initialization and keystream generation, the large state and long period length, that both the update and output functions are nonlinear, and that it passes standard statistical tests, MaD2 is unlikely vulnerable to distinguishing attacks.

4 Statistical Testing

A couple of statistical testing tools have been developed, among which are the most widely used NIST statistical test suite [7] and Diehard battery of tests [8]. A more stringent statistical test suite is TestU01 [10]. We tested our keystream generation using these three statistical test suites and the results are summarized in this section.

4.1 NIST Statistical Test Suite

The NIST tests are based on hypothesis testing. Each test is formulated to test a specific null hypothesis, i.e., a specific sequence of zeroes and ones is random. A probability value (*P-value*) is computed for each test, which summarizes the strength of the evidence against the null hypothesis. The probability that the null hypothesis for randomness is rejected for a random sequence is called the level of significance (α) of the test. If *P-value* $\geq \alpha$, then the null hypothesis is accepted; i.e., the sequence appears to be random. If *P-value* $< \alpha$, then the null hypothesis is rejected; i.e., the sequence appears to be non-random. Typically, α is chosen in the range [0.001, 0.01]. The NIST statistical test suite contains 15 tests (with some of them containing multiple sub-tests and generating multiple *P-value*s).

The empirical results can be interpreted in different ways. The two approaches adopted by NIST are examining the proportion of sequences that pass a statistical test and checking the distribution of *P-values* for uniformity. The first approach is reasonably accurate for large sample sizes (e.g., ≥ 1000 sequences). The second approach needs at least 55 sequences to provide statistically meaningful results.

We tested 1000 keystream sequences, each containing one million bits (125 KB). The significance level (α) is set to 0.01 in all tests. A random key is generated for each sequence and used to initialize the cipher. This random key can be up to 64 bytes and is generated from a modified RC4 using clock value as the input key. The only difference between the modified RC4 and the original RC4 is that the modified one discards the first 512 bytes of the pseudo-random output. MaD2 passed all the NIST statistical tests.

4.2 Diehard Battery of Tests

Most of the tests in Diehard return a *P-value*, which should be uniform on [0,1) if the input file contains truly independent random bits. A *P-value* near 0 or 1 indicates deviation from true randomness. This is in contrast with NIST tests, where a bigger *P-value* indicates better randomness. Some of Diehard tests yield more than one *P-value*, in which case a Kolmogorov–Smirnov (KS) test might be run on those *P-values* to produce a single *P-value* that indicates randomness [24]. The new Diehard release contains 17 tests, including some "tough" tests [9].

The C language implementation of Overlapping Sums Test is not a faithful interpretation of the author's original Fortran language implementation and none

of the (pseudo-)random number generators we tested can pass this test (please see the web page at`http://www.varioustopics.com/cryptography/782655-diehards-overlapping-sum-test.html` for more details). So we exclude this test from our testing. The remaining 16 tests are divided into two groups based on the minimum random sequence size that is needed by each test. GCD, Gorilla, and Overlapping Permutations need a much longer random sequence than other tests and are put in a group. All other tests are put in another group. For GCD, Gorilla, and Overlapping Permutations, we tested 50 random sequences, each containing 2176 million bits (272 MB). For other tests, we tested 100 random sequences, each containing 96 million bits (12 MB). Using this setup, at least 100 *P-values* are generated for each test. MaD2 passed all the tests except the Birthday Spacings test. To find the problem, we tested another three (pseudo-)random number generators: RC4, SHA1 (running in counter mode), and QRNG (a quantum random number generator available at `http://qrng.physik.hu-berlin.de/`). All of them failed this test too, which suggests it is more likely a problem with the test itself.

4.3 TestU01 Batteries of Tests

TestU01 is the most comprehensive statistical test suite that is publically available so far. It is a software library implemented in the ANSI C language. It offers a collection of utilities for the empirical statistical testing of uniform random number generators. Six pre-defined batteries of tests are available in TestU01. They are SmallCrush, Crush, BigCrush, Rabbit, Alphabit, and BlockAlphabit. Any (pseudo-)random number generator that implements both the *GetU01 (void *param, void *state)* and *GetBits (void *param, void *state)* interfaces defined in the *unif01_Gen* struct of TestU01 can use all the 6 pre-defined batteries of tests.

TestU01 requires much more (pseudo-)random numbers than the NIST and Diehard suites. It takes more than 12 hours to run all the 6 batteries on our machine. MaD2 is implemented in C programming language and created as a *unif01_Gen* object. Both *GetU01()* and *GetBits()* interfaces are implemented so that all 6 batteries of tests can be applied. Built-in parameters are used for SmallCrush, Crush, and BigCrush. For Rabbit, Alphabit, and BlockAlphabit, the size of bit sequence is set to 32×10^9. MaD2 cleared all the 6 TestU01 batteries of tests.

5 Performance Testing

The speed testing results for MaD2 keystream generation are given in Table 1. The testing results for RC4 and HC-128 [3] (the fastest one among the four software-efficient finalists of eStream) are also included in the table for comparison. The testing is done for a software implementation using C programming language. The C implementation closely follows the pseudo code given in section 2. There are no special optimizations done at the source code level except

Table 1. Keystream Generation Speed (cycle/byte)

Generator	Keystream size (KB)					
	1	5	10	100	1000	10000
RC4	9.53	7.67	7.09	6.98	7.04	7.04
HC-128	55.21	13.27	7.96	3.58	3.15	3.11
MaD2 (32-bit)	51.45	12.16	7.83	3.46	2.99	2.98
MaD2 (64-bit)	42.08	9.05	5.07	1.30	0.93	0.91

that register variables are used to minimize memory access whenever possible. Most modern compilers are smart enough and know more about code generation than the developer [25]. Therefore we leave optimizations largely to the compiler.

Both 32-bit and 64-bit executables compiled using Microsoft Visual C/C++ Optimizing Compiler Version 16 with option /O2 (optimized for maximum speed) are tested. RC4 and HC-128 are not 64-bit algorithms, but their 64-bit executables seem running slightly faster on our machine (Intel Core i3 370M, 2.4GHz, 64 KB L1 data cache, 64 KB L1 instruction cache, 512 KB L2 cache) and are chosen for the testing.

For each sequence size, we run each executable 30 times and get the average value of the top 3 speeds. The reason we exclude low speeds in our calculation is that the measured cycles may contain contributions from some system processes that we cannot stop and the small cycles more likely reflect the actual performance. The data given here are more for relative comparison than for benchmarking, which would require more comprehensive testing on different platforms. For the same reason, we refrain from using the commercial Intel C/C++ compiler, which has the potential to generate faster executables than Microsoft Visual C/C++ compiler on Intel platforms.

The state initialization of MaD2 takes about 38000 cycles, which is about 10 times slower than that of RC4. For this reason, RC4 beats MaD2 for short keystreams. For a 10 KB keystream, the 64-bit MaD2 executable outperforms RC4, which in turn outperforms the 32-bit MaD2 executable. For long keystreams (1 MB or more), the 32-bit MaD2 executable is more than 2 times faster than RC4 and the 64-bit MaD2 executable runs into one clock cycle per byte and is more than 7 times faster than RC4. Both the 32-bit and 64-bit MaD2 executables outperform HC-128 in all cases. For sequences of 1 MB or more, the 64-bit MaD2 executable is more than 3 times faster than HC-128.

6 Conclusion

In this paper we have presented a new word-based stream cipher, which can be efficiently implemented in software and is most efficient on 64-bit processors. On a typical Intel Core i3 personal computer, the cipher can run into one clock

cycle per byte, which is several times faster than any existing software ciphers we know. This makes it an appealing candidate for pervasive data encryption. It has a huge internal state of 8448 bits and is secure against various known attacks, although, as a new cipher, it is still subject to extensive cryptanalysis. We have also tested the pseudo-random number generation function of our cipher using the NIST suite, Diehard suite, and TestU01 suite. The testing did not raise any red flag.

References

1. FIPS Pub 197, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197 (2001)
2. Schneier, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn. John Wiley & Sons (1995)
3. Wu, H.: The Stream Cipher HC-128, `http://www.ecrypt.eu.org/stream/hcpf.html`
4. Boesgaard, M., et al.: The Stream Cipher Rabbit. eSTREAM report 2005/024 (2005), `http://www.ecrypt.eu.org/stream/papers.html`
5. Bernstein, D.J.: Salsa20/8 and Salsa20/12. eSTREAM report 2006/007 (2006), `http://www.ecrypt.eu.org/stream/papers.html`
6. Berbain, C., et al.: Sosemanuk, a fast software-oriented stream cipher. eSTREAM report 2005/027 (2005), `http://www.ecrypt.eu.org/stream/papers.html`
7. Runkin, A., et al.: Statistical Test Suite for Random and Pseudo Random Number Generators for Cryptographic Applications. NIST special publication 800-22
8. Marsaglia, G.: DIEHARD Battery of Tests. New version, `http://www.csis.hku.hk/~diehard/`
9. Marsaglia, G., Tsang, W.: Some Difficult-to-Pass Tests of Randomness. Journal Statistical Software 7(3) (2002)
10. L'Ecuyer, P., Simard, R.J.: Testu01: A C Library for Empirical Testing of Random Number Generators. ACM Trans. Math. Softw. 33(4) (2007)
11. Fog, A.: Instruction tables – Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs (2012), `http://www.agner.org/optimize/instruction_tables.pdf`
12. Roos, A.: A Class of Weak Keys in the RC4 Stream Cipher. Posting to sci.crypt (1995)
13. Golić, J.D.: Linear Statistical Weakness of Alleged RC4 Keystream Generator. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 226–238. Springer, Heidelberg (1997)
14. Fluhrer, S., Mantin, I., Shamir, A.: Weaknesses in the Key Scheduling Algorithm of RC4. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 1–24. Springer, Heidelberg (2001)
15. Klein, A.: Attacks on the RC4 Stream Cipher. Designs, Codes and Cryptography 48, 269–286 (2008)
16. Tews, E., Weinmann, R.P., Pyshkin, A.: Breaking 104 Bit WEP in Less Than 60 Seconds. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 188–202. Springer, Heidelberg (2008)
17. Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)

18. Hawkes, P., Rose, G.: Guess-and-Determine Attacks on SNOW. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 37–46. Springer, Heidelberg (2003)
19. Wong, K.K., Carter, G., Dawson, E.: An Analysis of the RC4 Family of Stream Ciphers against Algebraic Attacks. In: Proceedings of the Eighth Australasian Conference on Information Security, Brisbane, Australia, vol. 105, pp. 67–74 (2010) ISBN: 978-1-920682-86-6
20. Coppersmith, D., Halevi, S., Jutla, C.: Cryptanalysis of Stream Ciphers with Linear Masking. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 515–532. Springer, Heidelberg (2002)
21. Watanabe, D., Biryukov, A., De Cannière, C.: A Distinguishing Attack of SNOW 2.0 with Linear Masking Method. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 222–233. Springer, Heidelberg (2004)
22. Nyberg, K., Wallén, J.: Improved Linear Distinguishers for SNOW 2.0. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 144–162. Springer, Heidelberg (2006)
23. Lee, J.-K., Lee, D.-H., Park, S.: Cryptanalysis of Sosemanuk and SNOW 2.0 Using Linear Masks. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 524–538. Springer, Heidelberg (2008)
24. Soong, T.T.: Fundamentals of Probability and Statistics for Engineers, p. 327. John-Wiley and Sons Ltd. (2004) ISBN: 0470868147
25. Leitner, F.V.: Source Code Optimization, `http://www.linux-kongress.org/2009/slides/compiler_survey_felix_von_leitner.pdf`

Proofs of Retrievability via Fountain Code

Sumanta Sarkar and Reihaneh Safavi-Naini

Department of Computer Science, University of Calgary, Canada
{sarkas,rei}@ucalgary.ca

Abstract. Proofs of Retrievability (PoR) allows a client (verifier) to store a file at an untrusted remote storage, and later be able to check the integrity of the file through an interactive challenge-response protocol. A challenge specifies a random subset of blocks and the response is a function of the challenged block. An unbounded-use PoR scheme allows an arbitrary number of challenge-response interactions. Efficient PoR schemes must minimize the communication complexity of the challenge-response protocol, the storage overhead and computation of response by the prover. The security of a PoR scheme is against an erasing adversary and by showing the existence of an extractor which can extract the file from the set of challenges and their corresponding correct responses.

In this paper, we modify the unbounded-use PoR scheme of Shacham and Waters (2008) such that the number of challenged data blocks in each round is determined by a probability distribution over a set of possible values. For the security parameter l, the average number of challenged blocks is $O(\log l)$, and so is smaller that the original scheme of Shacham and Waters, and in the worst case, is $O(l)$. The response to a challenge is obtained by XORing the challenged data blocks and so is very fast. We show that to ensure security the original verification method of Shacham and Waters must be slightly modified.

Keywords: Cloud storage, linear homomorphic authenticator, Proofs of Retrievability, Raptor code.

1 Introduction

Cloud storage allows stored data to be accessible from anywhere at any time, and provides an attractive solution for back up of valuable data.

Storing a file in cloud however means there is no guarantee that the file stays intact. A dishonest cloud may erase some portions of the file to reduce its storage cost. To check that the file is correctly stored, a client can trivially download the whole file and check if the calculated MAC value of the file matches with the previously stored one. However, this becomes infeasible if the file size is large. Juels and Kaliski [5] introduced Proofs of Retrievability (PoR) protocol which verifies the integrity of the stored file through an audit protocol. The client applies an erasure code on the file M and stores the encoded file M' in the cloud. The encoded file M' is divided into blocks. The erasure code is such that the file M can be recovered from a fraction ρ of blocks of the encoded file M'. Along

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 18–32, 2013.

with M', the client also stores some extra information $\Delta(M)$ which will be used in the verification. The client only keeps a short key locally which it uses for the verification. Later during the audit which is through an interactive challenge-response protocol, the client plays the role of a verifier and the cloud plays the role of the prover. The security of a PoR scheme is formalized by showing the existence of an extractor which retrieves the file with very high probability from an erasing adversary that can pass the challenge-response protocol with some reasonable probability. The extractor collects the correct responses and decodes the file M from them. The efficiency of a PoR system is measured in terms of,

1. The computational cost of preparing a file for storing in the cloud, and calculating the response,
2. Communication cost required during a challenge-response interaction and,
3. The extra storage (overhead) needed for storing the file M.

The cost of encoding of the file M and creating $\Delta(M)$ is only once, and so is a fixed cost. However, the challenge-response protocol is run many times and hence it is important to reduce its cost. The cloud may engage in multiple audits with different users and so more efficient computation of response will improve its performance. Small size challenge improves the communication cost of the protocol, and also the computation cost of the prover as less blocks will be involved in the computation of response. The system in practice needs an unbounded number of challenge-response interactions (*unbounded-use*). In a *public verifiable* schemes, anyone knowing the appropriate public key can perform the verification. The system is called *private verifiable* when only the owner of the file who stores the file can run the challenge-response protocol.

In this paper, we are interested in private verifiable PoR schemes and present an unbounded-use scheme that improves the cost of response computation and the cost of communication of challenges in the average case.

The construction is based on the the construction of Shacham and Waters [10], and uses Fountain codes for challenge and response and file retrieval. Fountain codes are a special class of erasure codes which can generate an unbounded number of encoded symbols and have fast encoding and decoding algorithm [6,11]. We present a PoR scheme where the responses are the encoded symbols of a Raptor code [11] which is a special class of Fountain codes. Our scheme uses the probabilistic generation of encoded symbols in the Raptor code to determine the number of blocks to be challenged (and the expected response), and thus differs from all existing PoR schemes, where a fixed number of blocks are challenged in the challenge-response protocol.

1.1 Related Work

Proofs of Retrievability (PoR) was introduced by Juels and Kaliski (JK) [5] and subsequently has been extended and improved [10,2,3]. Proof of Storage (PoS) introduced by Naor and Rothblum [8] and Provable Data Possession (PDP) introduced by Ateniese et al. [1] are two closely related models for checking

the integrity of the file stored in the cloud, The details of PoR, PoS and PDP models differ, however they have the same insight: in a secure system if the cloud's response is verified, then there will be an extractor and by interacting with the cloud client will be able to retrieve the file with very high probability.

Juels and Kaliski (JK) [5] proved the NR scheme is secure in the model of Proofs of Retrievability and has quadratic communication complexity (in terms of security parameter) for response. This was improved to linear complexity by Shacham and Waters (SW) [10]. They achieved this by using homomorphic linear authenticators following a similar approach of Ateniese et al. [1]. Such authenticators aggregate the authenticators of individual file blocks and form a response that has the same size as that of a single authenticator. Shacham and Waters constructed both private and public verifiable unbounded-use PoR scheme, their public verifiable scheme are in the Random oracle model. Without using the random oracle, i.e., in the standard model, Dodis et al. [3] achieved the best known communication complexity for sending a challenge, which is linear in terms of the security parameter. They studied the PoR schemes explicitly using a coding theory approach. They viewed the set of all correct responses corresponding to the file M' stored in the cloud as a codeword C which is a *challenge-response* encoding of M'. The set of all responses for the same file M' from the prover form a word C' which may differ from C. The extractor decodes M from C'. This view of PoR enables us to choose the challenge-response encoding suitably in order to present improved PoR schemes.

Shacham and Waters Private PoR

We give an overview of the private PoR construction of [10] as our construction is based on that.

Storage(Encoding)

1. The client encodes the file M using an erasure code and obtains the encoded file M'.
2. The encoded file is divided into n blocks $m_1, \ldots, m_n$; each m_i is an element of $\mathbb{Z}_p$ and p is a prime.
3. A random number θ is chosen from $\mathbb{Z}_p$.
4. Corresponding to each block m_i, an authenticator σ_i is created using a pseudorandom function (PRF) f_{key} as

$$\sigma_i = f_{key}(i) + \theta m_i$$

5. The file blocks $\{m_i\}$ and $\{\sigma_i\}$ are stored in the cloud.

Audit

1. In the audit protocol, the client (verifier) chooses w random indices i from $\{1, \ldots, n\}$ and w random coefficients ν_i from $\mathbb{Z}_p$ and sends $Q = \{(i, \nu_i)\}$ to the prover.

2. The prover calculates

$$\sigma = \sum_{(i,\nu_i)\in Q} \nu_i \sigma_i \quad \text{and} \quad r = \sum_{(i,\nu_i)\in Q} \nu_i m_i. \tag{1}$$

   The pair (r, σ) is sent as a response.
3. After receiving the response, the verifier checks if the response is correct as follows

$$\sigma \stackrel{?}{=} \sum_{(i,\nu_i)\in Q} \nu_i f_{key}(i) + \theta r.$$

The novelty of this scheme is that the authenticators are homomorphic, which enables the verifier to check the integrity of multiple blocks using a single interaction.

Erasure Codes, Fountain Codes

An $[n, k]$ erasure code is a linear code which encodes k-message symbols into n coded symbols such that the k message symbols can be recovered from a subset of n codeword symbols, as long as the number of erased coded symbols is below a threshold that is determined by the minimum distance of the code. Reed Solomon codes are optimal erasure code in the sense that the k-symbol message can be recovered from any k coded symbols. That is $(n - k)$ symbols of the codeword can be erased, while maintaining recovery of the message.

Using block codes such as RS-codes for erasure decoding requires one to estimate the erasure rate (probability) of the file in advance in order to choose the rate $R = k/n$ of the code. In Fountain codes the sender generates potentially a limitless string of encoded symbols. The receiver can recover the message from sufficiently many encoded symbols. The decoding is usually done using belief propagation technique [4]. LT code by Luby [6] and Raptor codes by Shokrollahi [11] are two important classes of Fountain codes. The Raptor code has two stages of encoding. First, k message symbols $(x_1, \ldots, x_k)$ are encoded to $(y_1, \ldots, y_n)$ by an $[n, k]$ erasure code. Then the Fountain codeword symbols are generated by an LT code using linear combinations of symbols chosen randomly from $\{y_1, \ldots, y_n\}$. Raptor codes have linear encoding and decoding complexity. A good survey of Fountain codes is in [7].

1.2 Our Contributions

We consider PoR with private verification. We present a PoR scheme that uses the linear homomorphic authenticators and encoding of [10] but replaces the challenge and response system by a new system that is based on Fountain codes. The scheme supports unbounded number of challenge-response interactions between the verifier and the prover. The extractor uses the decoding algorithm of Fountain codes to decode the original file.

We consider the file blocks to be elements of $\mathbb{F}_2^l = \{0, 1\}^l$, where l is the security parameter, and the challenge coefficients to be always 1. A challenge set

simply includes a random set of indices $\{i_1, \ldots, i_w\}$. The response is by XOR of $w-1$ binary blocks of size l each, and so the prover only needs $w-1$ block XORs to compute a response. This requires $O(wl)$ steps. However, the prover computation (1) in SW PoR needs w multiplications over $\mathbb{Z}_p$, where p is an l-bit prime, followed by addition. This requires $O(wl^2)$ steps for the multiplication and $O(wl)$ steps for the addition. This means that the response computation in our case will be much faster.

However, in this setup, that is when challenge coefficients are all 1, [10] describes an attack (irrespective of XOR or modular addition), in which prover does not store the file in its actual form but still responds correctly with high probability. To avoid this attack we introduce an extra verification step. Details are given in Section 3.

In our scheme, the size of the challenge set w is not fixed: it is chosen probabilistically according to the degree distribution that is used in the encoding of the Raptor code.

We show that the size of a challenge set, on average is $O(\log l)$ and, in the worst case is $O(l)$. This means that to compute a response, the prover must process on average $O(\log l)$ blocks and in the worst case, $O(l)$ blocks. All existing unbounded-use PoR schemes have fixed size challenge of size $O(l)$.

2 Background

For the security model we follow [10] and use l as the security parameter. A PoR scheme consists of four algorithms **Kg**, **St**, $\mathcal{P}$ and $\mathcal{V}$.

- **Kg():** This randomized algorithm generates a secret key sk and the public key pk.
- **St(sk,M):** This randomized algorithm takes the secret key sk and the client file $M \in \{0,1\}^*$. Then it processes M and outputs M^* which is stored in the cloud. It also produces a tag t. The tag contains informations about the file and additional secret information encrypted under using a symmetric encryption system.
- $\mathcal{P}, \mathcal{V}$: The randomized algorithm that correspond to the prover and the verifier, respectively. During the protocol execution, both algorithms have as input, t that is output by St. The prover also takes the processed file M^* as input and the verifier takes sk as the input. At the end of the protocol, the verifier $\mathcal{V}$ outputs 0 or 1, where 1 means the file is being stored on the cloud. Precisely,

$$\{0,1\} \xleftarrow{R} (\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{P}(pk, t, M^*)).$$

We require the PoR to be correct and sound. Correctness means that if the prover is honest then

$$(\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{P}(pk, t, M^*)) = 1.$$

A PoR is *sound* if any prover that convinces the verification means that it actually holds the file.

Consider the following setup game between the adversary $\mathcal{A}$ and the environment:

1. The environment generates a key pair (pk, sk) by running $\mathbf{Kg}$ and provides pk to $\mathcal{A}$.
2. The adversary can now interact with the environment. It can make queries to a store oracle, providing, for each query, a file M. The environment computes $(M^*, t) \leftarrow \mathbf{St}(sk, M)$ and returns both M^* and t to the adversary.
3. For any M on which it previously made a store query, the adversary can undertake executions of the PoR protocol, by specifying the corresponding tag t. In these protocol executions, the environment plays the part of the verifier and the adversary plays the role of the prover: $\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{A}$. When a protocol execution completes, the adversary is provided with the output of $\mathcal{V}$. These protocol executions can be arbitrarily interleaved with each other and with the store queries described above.
4. Finally, the adversary outputs a challenge tag t returned from some store query, and the description of a prover $\mathcal{P}'$.

Definition 1. *The prover $\mathcal{P}'$ is ϵ-admissible if it convincingly answers an ϵ fraction of challenges, i.e., if*

$$Pr[(\mathcal{V}(pk, sk, t) \rightleftharpoons \mathcal{P}') = 1] \geq \epsilon.$$

Now we define the PoR.

Definition 2. *A PoR scheme is ϵ-sound if there exists an extraction algorithm such that, for every adversary $\mathcal{A}$, whenever $\mathcal{A}$ playing the setup game, outputs an ϵ-admissible prover $\mathcal{P}'$ for a file M, the extraction algorithm recovers M from $\mathcal{P}'$, except with negligible probability.*

2.1 Raptor Code

Suppose the message is $(x_1, \ldots, x_k)$, where each x_i is of l-bits, is to be transmitted through an erasure channel. First $(x_1, \ldots, x_k)$ is encoded to $(y_1, \ldots, y_n)$ by an erasure code C_n which can recover $(x_1, \ldots, x_k)$ from any ρn number of symbols. Symbols $y_1, \ldots, y_n$ are called input symbols. This step of encoding is called *precoding*. Next another round of encoding is applied to the n symbols $\{y_1, \ldots, y_n\}$ by a fountain code, in particular an LT code [6]. To encode n symbols using an *LT* code, a degree distribution is defined using a polynomial

$$w(x) = \sum_{i=1}^{n} w_i x^i$$

where w_i is the probability of choosing i, $i \in \{1, \ldots, n\}$. An encoded symbol is produced from the symbols $y_1, \ldots, y_n$ using the LT code as follows:

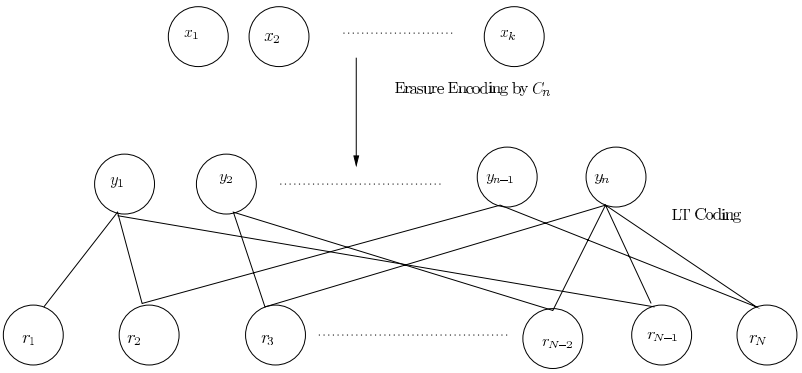


Fig. 1. Raptor Code structure

- Randomly choose a degree, say j, using $w(x)$.
- Choose uniformly at random, j symbols from the set $\{y_1, \ldots, y_n\}$, and XOR them to produce the encoded symbol

$$r_i = y_{i_1} \oplus \ldots \oplus y_{i_j}.$$

Suppose $r_1, \ldots, r_N$ are received at the receiver's end. These are called output symbols. If we think of a bipartite graph as given in Figure 1 with nodes $y_1, \ldots, y_n$ on one side and nodes $r_1, \ldots, r_N$ on the other side, then r_i is connected to the nodes $y_{i_1}, \ldots, y_{i_j}$ if and only if $r_i = y_{i_1} \oplus \ldots \oplus y_{i_j}$. The decoding of LT code is by using belief propagation whose success depends on the sparsity of this graph. The decoding can be described by the following example where symbols are over $\mathbb{F}_2$. Suppose source has symbols y_1, y_2, y_3 and the received symbols are $r_1 = 1, r_2 = 0, r_3 = 1, r_4 = 1$.

$$\begin{aligned} y_1 &= 1 \\ y_1 + y_2 + y_3 &= 0 \\ y_2 + y_3 &= 1 \\ y_1 + y_2 &= 1. \end{aligned}$$

So in the graph, degree of r_1 is 1, degree of r_2 is 3, degree of r_3 is 2 and the degree of r_4 is 2.

First equation gives $y_1 = 1$; pass the value of y_1 to the other equations to find,

$$\begin{aligned} y_2 + y_3 &= 1 \\ y_2 + y_3 &= 1 \\ y_2 &= 0. \end{aligned}$$

The third equation gives $y_2 = 0$; pass the value of y_2 to the other equations.

$$y_3 = 1$$
$$y_3 = 1.$$

Finally $y_3 = 1$. Now it is clear that to start this decoding one needs to have one node r_i with degree 1 and at each iteration, passing the value of recovered node y_i, means reducing by 1 the degree of each r_i to which y_i is connected by an edge. The decoding continues if there is an r_i with degree 1. Otherwise it halts.

After collecting $k+e$, for $e > 0$, symbols r_i, the receiver uses the LT decoder to recover ρn symbols of $\{y_1, \ldots, y_n\}$. The receiver can then apply erasure decoding of C_n to recover the message symbols $x_1, \ldots, x_k$.

From the decoding process, it is clear that for low decoding complexity the graph needs to be sparse. This means average number of edges between y_i nodes and r_i nodes should be small. The average number of edges is obtained as the mean of the distribution $w(x)$, i.e., $w'(1)$, where $w'(x)$ is the derivative of $w(x)$. The following results are from the Raptor code construction given in [11].

Let $\alpha > 0$ be a real number, set $D = \lceil 4(1+\alpha)/\alpha \rceil$ and define

$$w_D(x) = \frac{1}{\mu + 1}(\mu x + \sum_{i=2}^{D} \frac{x^i}{(i-1)i} + \frac{x^{D+1}}{D}), \tag{2}$$

where $\mu = (\alpha/2) + (\alpha/2)^2$. The average of w_D is

$$\ln(1/\alpha) + \beta + O(\alpha), \tag{3}$$

where $1 < \beta < 1 + \gamma + \ln(9)$, the constant γ is the Euler's constant.

Lemma 1. *[11, Lemma 4] There exists a positive real number c (depending on α) such that with an error probability of at most e^{-cn} any set of $(1+\alpha/2)n+1$ output symbols of the LT-code with distribution w_D and n-input symbols $y_1, \ldots, y_n$ are sufficient to recover at least ρn input symbols from $\{y_1, \ldots, y_n\}$ via belief propagation decoding, where $\rho = 1 - \frac{\alpha/4}{1+\alpha}$.*

Using this lemma we get the following result.

Theorem 1. *[11, Theorem 5] Let $\alpha > 0$ be a real number, k an integer, $D = \lceil 4(1+\alpha)/\alpha \rceil$, $R = (1+\alpha/2)/(1+\alpha)$, $n = \lceil k/R \rceil$. Let C_n be an erasure code which can decode $(1-R)/2$ erasures. Then the Raptor code with precode C_n and the LT-code with the distribution $w_D(x)$ which encodes k symbols, can decode from $(1+\alpha)k$ output symbols.*

Proof. From Lemma 1 one needs $(1+\alpha/2)n + 1$ output symbols r_i to get ρn symbols of $\{y_1, \ldots, y_n\}$ with error probability e^{-cn}. Note that replacing $n = \lceil k/R \rceil$ we get $(1+\alpha/2)n + 1 = (1+\alpha)k + 1 \approx (1+\alpha)k$. In this case $\rho = 1 - (1-R)/2$. So from ρn symbols of $\{y_1, \ldots, y_n\}$ one can decode the message symbols $x_1, \ldots, x_k$ by using the erasure decoding of C_n. □

3 A PoR Scheme Based on Raptor Codes

In this section, we discuss PoR construction based on the Raptor code. Dodis et al. [3] viewed the set of all possible correct responses corresponding to the encoded file M' as a codeword C, which they call a challenge-response encoding of M'. The set of all responses received from the prover for the a file M', form a word C' which may differ from C. The job of the extractor is to decode M from C'. We follow this view and use LT code as the encoding of the challenge and response. Thus after encoding file M by an erasure code and storing the encoded file M' in the cloud, the responses are the encoded symbols of the Raptor code applied to M. In fact, the whole structure of this PoR matches the Raptor code, where the message file is encoded by a precode and after that streams of encoded symbols are generated by the LT code. To have a PoR, it only remains to have a verification algorithm which verifies that a response (encoded symbol of the LT code) is correct. Bowers et al. [2] also had the same view, where the responses are formed by the encoding of a code which they call "inner code". However, their scheme only supports bounded number of challenges as the challenge is based on the precomputed challenge-response pairs. By using Raptor code, we can directly invoke the decoding analysis of that code to analyze the extractor of the PoR. It is to be noted that the notion of the extractor comes in the security proof. Therefore, extractor does not need to be very efficient. This is it is sufficient to be able to decode the file with negligible decoding failure probability.

3.1 The Construction

We use the same homomorphic authenticators as [10] for the verification. We construct the Raptor-PoR protocol for private verification as follows.

Raptor-PoR
Let the erasure probability of the cloud be $1-\rho$. Let the client file be $M = (x_1, \ldots, x_k)$, each x_i is an element of $\mathbb{F}_2^l$. For each x_i, the authenticator also has l-bits, so we choose l as the security parameter. The mapping $PRF : \{0,1\}^* \times \mathcal{K}_{prf} \rightarrow \{0,1\}^l$ is a pseudorandom function. In Raptor-PoR, the algorithm **kg** and **St** are run to process the file for the storage. In the audit, the algorithm $\mathcal{V}$ is run by the verifier and $\mathcal{P}$ is run by the prover. The algorithm $\mathcal{V}$ has three modules: $\mathcal{V}.Tagcheck$, $\mathcal{V}.Chal$ and $\mathcal{V}.Ver$.

Storage (Encoding)

- **Kg()**: A random symmetric encryption key $k_{enc} \xleftarrow{R} \mathcal{K}_{enc}$ and a random MAC key $k_{mac} \xleftarrow{R} \mathcal{K}_{mac}$ are chosen. The secret key is $sk = (k_{enc}, k_{mac})$. Since this is private verification, there is no public key pk.
- **St(sk,M)**: First $M = (x_1, \ldots, x_k)$ is encoded by an erasure code C_n to obtain $M' = (y_1, \ldots, y_n)$, where C_n is such that any ρn symbols from $(y_1, \ldots, y_n)$ will be enough for the reconstruction of M. Now choose a PRF key $k_{prf} \xleftarrow{R} \mathcal{K}_{prf}$ and a random binary $l \times l$ matrix $A = [A_1, \ldots, A_l]^T$, where

each A_i is an l-bit row vector.
Let $t_0 = n||Enc_{k_{enc}}(k_{prf}||A_1||\cdots||A_l)$, and $t = t_0||MAC_{k_{mac}}(t_0)$ be the file tag.
For each i, where $1 \leq i \leq n$, create authenticators $\sigma_1, \ldots, \sigma_n$ as

$$\sigma_i = PRF_{k_{prf}}(i) \oplus y_i A \qquad \text{for } 1 \leq i \leq n.$$

Each σ_i is also an l-bit symbol. Then $M^* = (y_1, \ldots, y_n, \sigma_1, \ldots, \sigma_n)$ is the processed file. Send M^* and t to the cloud.

Audit

- $\mathcal{V}.Tagcheck(sk, t)$: Obtains k_{mac} and k_{enc} from the secret key sk. Receives the tag t from the prover and verify it by the k_{mac}, if MAC does not match, quit the audit. Otherwise, using the symmetric key k_{enc}, decrypt $Enc_{k_{enc}}(k_{prf}||A_1||\cdots||A_l)$ and recover n, k_{prf} and the matrix A.
- $\mathcal{V}.Chal(n)$: Choose an integer w using the degree distribution with the generator polynomial $w_D(x) = \sum_{i=1}^{n} w_i x^i$. Then choose w indices, say $\{i_1, \ldots, i_w\}$, uniformly from $\{1, \ldots, n\}$ and choose one index, say c, uniformly at random from $\{i_1, \ldots, i_w\}$. Send $Q = (\{i_1, \ldots, i_w\}, \{c\})$ to the prover.
- $\mathcal{P}(Q, M^*)$: In response to the challenge Q compute

$$\begin{aligned} r &= y_{i_1} \oplus \ldots \oplus y_{i_w} \\ \sigma &= \sigma_{i_1} \oplus \ldots \oplus \sigma_{i_w}. \end{aligned} \tag{4}$$

  Send $resp = (r, \sigma, y_c, \sigma_c)$ to the verifier.
- $\mathcal{V}.Ver(A, k_{prf}, resp)$: After receiving prover's response, check whether

$$\sigma \stackrel{?}{=} rA \oplus \sum_{i \in \{i_1, \ldots, i_w\}} PRF_{k_{prf}}(i),$$
$$\sigma_c \stackrel{?}{=} PRF_{k_{prf}}(c) \oplus y_c A.$$

3.2 Security Proofs

For the file retrieval, the client relies on the correct response value which is the linear sum of the challenged blocks. If the prover were only asked to send the value r and σ instead of $(r, \sigma, y_c, \sigma_c)$, the prover could pass the verification process without storing the actual file, with very high probability. This kind of attack has been shown in [10, Appendix B]. We discuss the attack in Appendix A which shows the cloud can store the file in a different form but still can pass the verification, i.e., provides correct response value, with probability 1/2. To avoid this attack, the verifier introduces an extra layer of verification in which the prover also sends y_c and its corresponding authenticator σ_c.

Theorem 2. *Suppose the MAC is unforgeable, the symmetric encryption scheme is semantically secure and the PRF is secure. Then, the following is true with an overwhelming probability: no adversary against the soundness of the proposed*

PoR scheme, will be able to pass the verification test in a challenge-response instance, except by responding correct values $(r, \sigma, y_c, \sigma_c)$ *which is the response computed by an honest prover.*

Proof. Applying the proof of [10, Theorem 4.1], it can be shown that if the response values r and σ pass the verification, i.e., they comply with the relation $\sigma = rA \oplus \sum_{i\in\{i_1,\ldots,i_w\}} PRF_{k_{prf}}(i)$, then r is the correct sum of $y_{i_1}, \ldots, y_{i_w}$. The proof relies on the security of the MAC, the symmetric encryption and the PRF, which guarantee that a dishonest prover can not forge the authenticators σ_i's. The proof idea is the following. The authenticator σ_i for the data block y_i cannot be forged if the matrix A and the PRF key k_{prf} are kept secret. Note that k_{prf} and A are stored in the cloud in the encrypted form using a semantically secure symmetric encryption. Further authenticity of the the encrypted value is maintained by a secure MAC.

However, in this setup, a correct value of r, does not guarantee that r is computed by summing the actual values of $y_{i_1}, \ldots, y_{i_w}$. Some different values $y'_{i_1}, \ldots, y'_{i_w}$ may lead to the same sum r, i.e. $r = y_{i_1} \oplus \cdots \oplus y_{i_w} = y'_{i_1} \oplus \cdots \oplus y'_{i_w}$. It can be seen in Appendix A, how a dishonest prover can compute r correctly with nonnegligible probability without storing the actual file. Suppose a dishonest prover computes r correctly but with different $y_{i_1}, \ldots, y_{i_w}$ with probability ϕ. If that prover does not have the actual value of y_c, then to pass the second step of verification, i.e., $\sigma_c = PRF_{k_{prf}}(c) \oplus y_c A$, it has to forge the authenticator σ_c. But the authenticator is proved to be secure. Therefore, the dishonest prover has no choice but to guess y_c for which σ_c is the authenticator. In this case the prover passes this step of verification with probability $\frac{1}{2^l}$ (since, y_c is an l-bit element). Thus the dishonest prover passes the two steps of verification with probability $\phi\frac{1}{2^l} < \frac{1}{2^l}$, which is negligible. □

We use the same erasure code C_n and degree distribution $w_D(x)$ as that of the Raptor code to construct the PoR scheme. The retrieval of the file is simply the decoding process of the Raptor code. Referring to the parameters of the Raptor code described in Section 2.1, the extractor needs $(1+\alpha)k$ output symbols for the retrieval of the file. The client file has k blocks each of size l-bits, where $k = poly(l)$. Below we express different parameters of the Raptor code in terms of l.

1. Take $\alpha = 1/l$.
2. Rate of the precode C_n is $R = \frac{1+\alpha/2}{1+\alpha} = \frac{2l+1}{2l+2}$. Then $n = k/R = k\frac{2l+2}{2l+1} = poly(l)$, since $k = poly(l)$.
3. Choose C_n such that the erasure probability that it can handle is $1 - \rho = (1 - R)/2 = \frac{1}{4(l+1)}$.
4. For LT code: $D = 4(l + 1)$, $\mu = \frac{1}{2l} + \frac{1}{4l^2}$, the degree distribution is

$$w_D(x) = \frac{2l+1}{4l^2+2l+1}x + \frac{4l^2}{2l+1}\left(\frac{x^2}{1.2} + \frac{x^3}{2.3} + \ldots + \frac{x^{4(l+1)}}{(4l+3)(4l+4)} + \frac{x^{4l+5}}{4l+4}\right).$$

The mean of this distribution is $\ln(l)+\beta+O(1/l)$, where $1 < \beta < 1+\gamma+\ln(9)$, the constant γ is the Euler's constant.

Using these parameters, we can use the decoding result of the Raptor code (Theorem 1). Thus we have the following result on the file retrieval.

Theorem 3. *If the prover is ϵ-admissible then running the PoR protocol for $\frac{(1+1/l)k}{\epsilon}$ iterations, the extractor will be able to retrieve the file with error probability $e^{-poly(l)}$.*

Proof. When the extractor plays against an ϵ-admissible prover, then each output symbol is correct with probability ϵ. Therefore, to collect $(1 + 1/l)k$ output symbols the extractor should run $\frac{(1+1/l)k}{\epsilon}$ challenge-response interactions. By Theorem 1, the extractor needs $(1 + 1/l)k$ output symbols to recover ρ fractions of symbols of the precode C_n, where the error probability is $e^{-O(n)}$. Note that $n = poly(l)$ and so the error probability is $e^{-poly(l)}$, which is negligible. □

3.3 Cost Comparison

The three known unbounded PoR constructions are from Juels and Kaliski [5], Shacham and Waters [10], and Dodis et al. [3]. In all of these schemes, the cloud storage is $\frac{2|M|}{R}$, where $|M|$ is the size of the client file and R is the rate of the code that encodes M before storing it in the cloud. Clearly, the encoded file size is $|M|/R$. The factor 2 comes from the set of all authenticators. Our scheme has the same cloud storage cost, i.e., $\frac{2|M|}{R}$. The precode, which encodes the file M, has rate $R = \frac{2l+1}{2l+2}$. Thus cloud storage is $\frac{2(2l+2)|M|}{2l+1}$.

For a challenge of size w, the response r is the XOR of w elements, and σ is the XOR of another w elements of $\mathbb{F}_2^l$. The response computation of [3] is the same as in [10]. We also use homomorphic linear authenticators and compare the cost of computing a response in our scheme with that of Shacham and Waters [10]. In [10], for a challenge of size w, response r and σ are computed (see (1)) by using w multiplications over $\mathbb{Z}_p$ and followed by the addition of $w - 1$ elements of $\mathbb{Z}_p$, where p is an l-bit prime. The multiplication of two l-bit numbers over $\mathbb{Z}_p$ requires $O(l^2)$ steps and for addition it is $O(l)$. Therefore, for SW scheme, the response calculation in (1) requires $O(wl^2)$ steps. As we consider the file blocks over $\mathbb{F}_2^l$ and the challenge coefficients ν_i to be always 1, the computation does not require multiplication and response calculations only needs $(w-1)$, the same cost is required for computing σ, and thus requires $O(wl)$ steps in total. This means that the response computation is much faster in our case.

We note that all previous schemes challenge a fixed number of blocks of order $O(l)$. In our scheme, the number of challenged blocks is not fixed: for a security parameter l, the size of the challenge set is chosen from the interval $[1, 4l + 5]$ according to the probability distribution $w_D(x)$ with $\alpha = 1/l$. Thus the number of challenged blocks is $O(l)$ but, the size of the challenge set on average is the mean of the distribution $w_D(x)$. For $\alpha = 1/l$, the mean is $\ln(l) + \beta + O(1/l)$, where $1 < \beta < 1 + \gamma + \ln(9)$, the constant γ is the Euler's constant. Therefore on the average number of challenged indices is $O(\log l)$.

In our scheme, a challenge includes $O(\log l)$ indices in the average case and the size of the response is $O(l)$. So the total communication complexity is $O(l)$, which is the same for the existing unbounded PoR schemes [5,10,3].

Table 1. Response cost comparison between Raptor-PoR and PoR of Shacham and Waters [10]

	Computation of a Response for w-size challenge	Number of Challenged blocks
RAPTOR-PoR	$O(wl)$	$O(\log l)$ (Average)
SW PoR	$O(wl^2)$	$O(l)$

4 Conclusions

In this paper, we presented a PoR scheme, based on the Shacham-Waters scheme [10], that uses variable length challenges and has efficient response computation. Our construction uses ideas from fountain codes and unlike other PoR schemes, the size of challenge set is not fixed. The number of challenged blocks on average however is $O(\log l)$ which is less than other known schemes.

For implementation of the scheme in practice, we need to divide the file into several chunks of suitable size and apply encoding on each chunk separately. For example, one can divide the file into chunks each having 223 symbols of size of 1 byte each. Then use the $(255, 223)$ Reed-Solomon code over $GF(2^8)$ to encode each chunk, and finally obtain the encoded file by concatenating the encoded chunks. The encoded file is further divided into blocks each having l bits, where l is the security parameter (10 bytes, for instance), and authenticators of size l bits for each block is computed. Finally the encoded file and the authenticators are stored in the cloud. The challenge response will be as before but the decoding of the original file from the responses will need extra steps. From the correct responses the extractor needs to collect l-bit blocks which belongs to the encoding of the same chunk and decode the chunk when it has received enough symbols of that particular encoded chunk. The analysis of the extractor also will be slightly different.

An efficient implementation of this scheme will establish its importance in practice.

Acknowledgments. The authors would like to thank Alberta Innovates Technology Future which has partially supported this work. The authors are also thankful to Liangfeng Zhang for his valuable editorial comments.

References

1. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.X.: Provable data possession at untrusted stores. In: Ning, et al. (eds.) [9], pp. 598–609
2. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: theory and implementation. In: Sion, R., Song, D. (eds.) CCSW, pp. 43–54. ACM (2009)

3. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of Retrievability via Hardness Amplification. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 109–127. Springer, Heidelberg (2009)
4. Gallager, R.G.: Low-Density Parity-Check Codes. PhD thesis, MIT Press, Cambridge, MA (1963)
5. Juels, A., Kaliski Jr., B.S.: PORs: proofs of retrievability for large files. In: Ning, et al. (eds.) [9], pp. 584–597
6. Luby, M.: LT codes. In: FOCS, p. 271. IEEE Computer Society (2002)
7. Mackay, D.J.C.: Fountain codes. IEEE Communications 152, 1062–1068 (2005)
8. Naor, M., Rothblum, G.N.: The complexity of online memory checking. In: FOCS, pp. 573–584. IEEE Computer Society (2005)
9. Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.): Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31. ACM (2007)
10. Shacham, H., Waters, B.: Compact Proofs of Retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)
11. Shokrollahi, A.: Raptor codes. IEEE Transactions on Information Theory 52(6), 2551–2567 (2006)

A Verification without Storing the Actual File

Suppose, the cloud stores the file in a different way which is as follows. It selects randomly one block y_j from the encoded file $y_1, \ldots, y_n$ and replaces each y_i as $y_i = y_i \oplus a_i y_j$ for all $i \neq j$, where a_i is chosen randomly from $\{0, 1\}$. Then it removes y_j forever. In this situation cloud still may be able to pass the verification. Suppose, a challenge of w indices $\{i_1, \ldots, i_w\}$ sent by the client, which does not include j. Then the cloud computes the response as

$$\begin{aligned} R &= y_{i_1} \oplus a_{i_1} y_j \oplus \ldots \oplus y_{i_w} \oplus a_{i_w} y_j \\ &= r \oplus y_j (a_{i_1} \oplus \ldots \oplus a_{i_w}). \end{aligned}$$

The other response, i.e., the sum of the authenticators will be

$$\sigma = \sigma_{i_1} \oplus \ldots \oplus \sigma_{i_w}.$$

Note that this pair (R, σ) will verify if and only if $R = r$. Now $R = r$ holds if $a_{i_1} \oplus \ldots \oplus a_{i_w} = 0$. Suppose, w is even. Then this happens if the weight of the string $(a_{i_1}, \ldots, a_{i_w})$ is even which can happen in $\binom{w}{2} + \ldots + \binom{w}{w}$ many ways. Then the probability

$$\begin{aligned} P(a_{i_1} \oplus \ldots \oplus a_{i_w} = 0) &= \frac{\binom{w}{2} + \ldots + \binom{w}{w}}{2^w} \\ &\approx \frac{2^{w-1}}{2^w} \\ &= \frac{1}{2}. \end{aligned}$$

On the other hand if the challenge $\{i_1, \ldots, i_w\}$ contains the index j, suppose $i_w = j$. Then the cloud computes the response as

$$\begin{aligned} R &= y_{i_1} \oplus a_{i_1} y_j \oplus \ldots \oplus y_{i_{w-1}} \oplus a_{i_{w-1}} y_j \\ &= y_{i_1} \oplus \ldots \oplus y_{i_{w-1}} \oplus y_j (a_{i_1} \oplus \ldots \oplus a_{i_w - 1}). \end{aligned}$$

The response from an honest cloud in this case would be $r = y_{i_1} \oplus \ldots \oplus y_{i_w - 1} \oplus y_j$. Therefore, $R = r$ is held if $a_{i_1} \oplus \ldots \oplus a_{i_w - 1} = 1$. This happens if the weight of the string $(a_{i_1}, \ldots, a_{i_w - 1})$ is odd and that can happen in $\binom{w-1}{1} + \ldots, \binom{w-1}{w-1}$ ways. Note that since we are considering w even, so $w - 1$ is odd. Therefore, the probability

$$\begin{aligned} P(a_{i_1} \oplus \ldots \oplus a_{i_w} = 1) &= \frac{\binom{w-1}{1} + \ldots + \binom{w-1}{w-1}}{2^{w-1}} \\ &= \frac{2^{w-2}}{2^{w-1}} \\ &= \frac{1}{2}. \end{aligned}$$

So we see that the cloud can pass the verification process with probability 1/2.

MARC: Modified ARC4

Jianliang Zheng[1] and Jie Li[2]

[1] Independent Scholar, New York, NY, USA
zheng@ee.ccny.cuny.edu
[2] Department of Computer Science, Graduate Center,
City University of New York, New York, NY, USA

Abstract. RC4, often referred to as Alleged RC4 (ARC4) in open literature, was and probably still is the most popular stream cipher. Although some weaknesses in its key scheduling algorithm have been reported and new faster and claimed secure stream ciphers have been proposed, ARC4 is likely to remain as a big player in cryptographic applications. In this paper, we propose a new variant of ARC4, called Modified ARC4 (MARC), which enhances the security of ARC4 by modifying its key scheduling algorithm and improves the performance by modifying its pseudo-random generation algorithm. MARC retains the simplicity of ARC4 and is faster than most software-efficient finalists of eStream.

Keywords: RC4, ARC4, MARC, Stream Cipher, High Performance.

1 Introduction

RC4 was designed by Ron Rivest in 1987 and kept as a trade secret of RSA Security until it leaked out in 1994. Because RSA Security has never officially released the algorithm and the name RC4 is trademarked, what is often referred to in the open literature is ARC4 (Alleged RC4) [1]. Due to the huge number of internal states, no practical attacks have been mounted on the internal state so far. However, RC4 has some weaknesses in its key scheduling algorithm [2]. This has been exploited to break the Wired Equivalent Privacy (WEP) encryption used within 802.11 wireless local area networks. RSA Security responded to this and recommended that "The initial key scheduling component of RC4 should for now be routinely amended for new applications to include hashing and/or discarding the first 256 bytes of pseudo-random output." [3]

In this paper, we propose a new variant of ARC4, called Modified ARC4 (MARC), which enhances the security of ARC4 by modifying its key scheduling algorithm and improves the performance by modifying its pseudo-random generation algorithm. MARC retains the simplicity of ARC4 and is faster than all the software-efficient finalists of eStream [4,5,6,7] except HC-128 [4]. It is the fastest one among the variants of ARC4 [8,9,10,11] we know.

The rest of this paper is organized as follows: Section 2 describes the algorithm details. Section 3 gives a security analysis of the algorithm. Section 4 and 5 provide the statistical testing results and performance testing results respectively. Finally section 6 concludes the paper.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 33–44, 2013.

Listing 1. ARC4

```
1  ## addition (+) and increment (++) operations ##
2  ## are performed modulo 256; % means modulo.  ##
3
4  # Key Scheduling Algorithm (KSA)
5  for i from 0 to 255
6      S[i] = i
7  endfor
8  j = 0
9  for i from 0 to 255
10     j = j + S[i] + key[i % keylength]
11     swap(S[i], S[j])
12 endfor
13
14 # Pseudo-Random Generation Algorithm (PRGA)
15 i = 0
16 j = 0
17 while GeneratingOutput
18     i++
19     j = j + S[i]
20     swap(S[i], S[j])
21     n = S[i] + S[j]
22     output S[n]
23 endwhile
```

2 Design

In this section we present the algorithm details. We first give a brief review of ARC4 [1] and then describe the differences between ARC4 and MARC.

2.1 ARC4

ARC4 is simple and ideal for software implementation. It maintains an internal state, which consists of a permutation of all $2^8 = 256$ possible octets and two 8-bit indices for accessing elements in the permutation. The permutation is first initialized to the identity permutation and then shuffled using a secret key according to the key scheduling algorithm (KSA) given on lines 5 to 12 of Listing 1. Afterwards, pseudo-random number sequence is generated using the pseudo-random generation algorithm (PRGA) given on lines 15 to 23 of Listing 1. The PRGA iterates as many times as needed and during each iteration it continues to shuffle the permutation by swapping two elements and outputs one octet. To encrypt or decrypt a message, simply act the pseudo-random number sequence on the message using bitwise XOR.

2.2 MARC

The KSA and PRGA of MARC are shown in Listing 2. The new KSA iterates 576 times instead 256 times to shuffle the identity permutation. In ARC4 the first *keylength* bytes of S are likely to have similar patterns after key scheduling if similar keys that only differ at the end are used. The additional 256 + 64 = 320 iterations (the maximum allowed *keylength* in MARC is 64) are used for breaking such patterns. Note that we use a new 16-bit unsigned integer r instead of the 8-bit index i as the loop counter in the second *for* loop. This is because we need to iterate 576 times but index i can only have 256 different values. Except r, all other variables are 8-bit unsigned integers. The new KSA introduces a third index k, which is initialized to 0 and then XORed with index j during

Listing 2. Modified ARC4 (MARC)

```
## addition (+) and increment (++) operations     ##
## are performed modulo 256; except variable r,  ##
## which is a 16-bit unsigned integer, all other ##
## variables are 8-bit unsigned integers;        ##
## ^ means bitwise XOR.                          ##

# Key Scheduling Algorithm (KSA)
for i from 0 to 255
    S[i] = i
endfor
i = 0
j = 0
k = 0
for r from 0 to 575
    j = j + S[i] + key[i % keylength]
    k = k ^ j
    left_rotate(S[i], S[j], S[k])
    i++
endfor

# Pseudo-Random Generation Algorithm (PRGA)
# (j and k are from KSA)
i = j + k
while GeneratingOutput
    i++
    j = j + S[i]
    k = k ^ j
    swap(S[i], S[j])
    m = S[j] + S[k]
    n = S[i] + S[j]
    output S[m]
    output S[n]
    output S[m ^ j]
    output S[n ^ k]
endwhile
```

each iteration. It also replaces ARC4's swap operation between $S[i]$ and $S[j]$ with a left rotation operation among $S[i]$, $S[j]$, and $S[k]$ (i.e., $tmp = S[i]$, $S[i] = S[j]$, $S[j] = S[k]$, $S[k] = tmp$) during each iteration.

The new PRGA keeps the values of index j and k from KSA and sets the initial value of index i to the sum of index j and k. It shuffles the permutation like ARC4 PRGA but outputs 4 octets instead of one octet during each iteration using index j, k, m, and n (m is the sum of $S[j]$ and $S[k]$; n is the sum of $S[i]$ and $S[j]$). The 4 octets are: $S[m]$, $S[n]$, $S[m \hat{} j]$, and $S[n \hat{} k]$. The combination of m, n, $m \hat{} j$, and $n \hat{} k$ renders excellent statistical properties as attested by the most stringent statistical testing tools (see section 4).

3 Security

In this section we discuss the security of MARC. We will not give a complete cryptanalysis here. Instead we will only focus on the differences between MARC and ARC4. For detailed cryptanalysis of ARC4, the reader is referred to [2,12,13,14,15].

The main weakness of ARC4 is its simple KSA. A simple remedy is to discard the first $n \times 256$ ($n = 1, 2, ...$) bytes of output, which in effect introduces more shuffling before generating pseudo-random numbers. The KSA of MARC iterates 576 times and each time it rotates three values. The KSA of ARC4 only iterates 256 times and each time it swaps two values. If one rotation is computed as 1.5 swaps, then MARC swaps 864 times. This is more than three times of ARC4. Note that this is not exactly the same as dropping 864 - 256 = 608 bytes at the beginning since the shuffling during pseudo-random number generation does not involve the key as key scheduling does.

For similar keys that only differ at the end, the differences do not come into play before the *keylength*-th ($keylength \leq 64$) iteration of key scheduling. This means the first *keylength* bytes are very likely to have a similar distribution for those keys. MARC deals with this by shuffling the first 64 bytes one more time than the other part. At the end of key scheduling, MARC also sets the value of index i to the sum of index j and k and persists all three indices for future use. Because the output sequence is highly sensitive to the values of the index i, j, and k, making indices depend on the secret key instead resetting them to 0 at the beginning of the PRGA also helps solve the correlation problem among similar keys.

To measure the differences between MARC KSA and ARC4 KSA, we performed avalanche testing for both of them. The testing steps are:

1. Randomly select a key of size 64 (worst case for diffusion), denoted as K1.
2. Get the following variants of K1:
   (a) K2 = 1's complement of K1
   (b) K3 = flip of K2 (left right flip)
   (c) K4 = 1's complement of K3

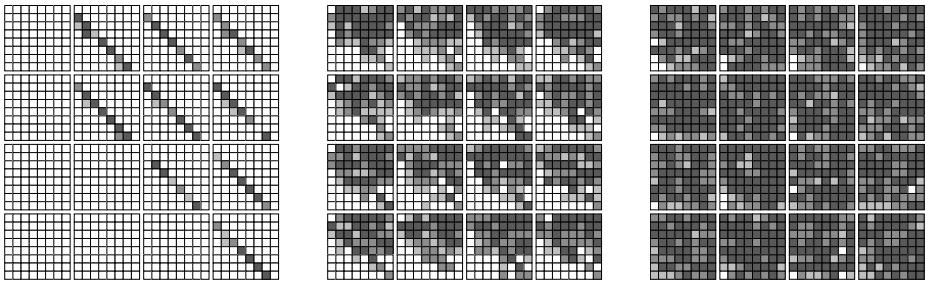
   If K2 and K3 are same (i.e., input is symmetric), then K3 and K4 are not used.

3. For each of the above K1, K2, K3, and K4, we flip one bit of it each time (starting from the most significant bit) and compare the initialized state with the unflipped version.
4. Repeat above steps until the number of flippings reaches the required number, which is 10^6 (the actual number of flippings may exceed 10^6, since we do not stop until all the bits of a key or its variant are flipped).

Ideally, for the flipping of each input bit we want the flipping of each output bit to be like the flipping of a coin, which is a binomial distribution with a probability $p = 0.5$ for each of the two possible outcomes. According to the central limit theorem, it can be approximated using normal distribution for a not too small sample size *szSample* with mean $\mu = szSample \times p$ and standard deviation $\sigma = \sqrt{szSample \times p \times (1-p)}$. For a normal distribution, an outcome has a probability of 68.3% to be $\mu \pm \sigma$, a probability of 95.4% to be $\mu \pm 2\sigma$, and a probability of 99.7% to be $\mu \pm 3\sigma$. For each output bit, we check if its flipping probability falls in the range $[\mu - n\sigma, \mu + n\sigma]$ ($n = 1, 2, 3$, checked in that order). The corresponding output bit gets n point(s) if the probability does fall in the range; it gets 4 points otherwise. These four different values, from the best 1 point to the worst 4 points, are depicted using different colors: dark gray (▪) for 1, gray (▪) for 2, light gray (▪) for 3, and white for 4.

The testing results for both ARC4 KSA and MARC KSA are shown in Fig. 1. Each small square plotted in one of the four different colors shows how well the flipping of an input bit (row) causes the flipping of an output bit (column). The input bit range is [1, 32]. The three output bit ranges are [1, 32], [257, 288], and [513, 544] respectively. ARC4 KSA has a really poor avalanche effect at the beginning of the internal state (Fig. 1 (a)). The testing revealed that the flipping probability is far below the ideal value 0.5 for those bits at the beginning. The situation improves as we move the output bit range away from the beginning of the internal state (Fig. 1 (b)). The avalanche effect is significantly better when the output bit range is shifted out of the first *keylength* (here 64) bytes of the internal state (Fig. 1 (c)). MARC KSA has a better avalanche effect than ARC4 KSA in all three cases.

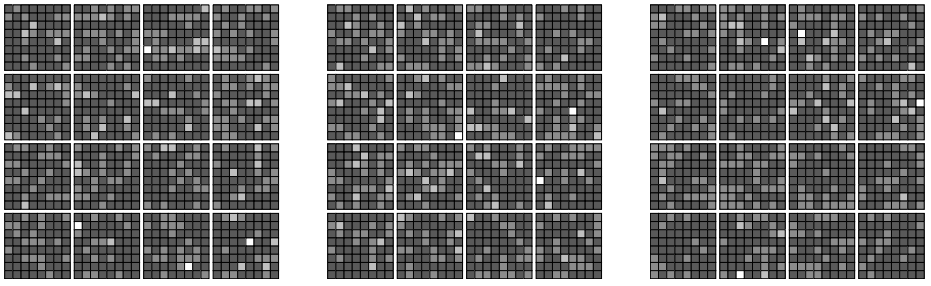
The internal state of ARC4 evolves relatively slowly and what matters more is the change of n if a short sequence (e.g., a few bytes) is to be generated. In this case, it makes no big difference whether one generates 4 bytes using 4 different values of index n or using 4 different indices at once, as long as the 4 values of n and the 4 different indices are both computed in a proper way. In ARC4, $n = S[i] + S[j]$. $S[i]$ iterates through the state table but does not guarantee each value is accessed exactly once due to the swap operation. $S[j]$ is more irregular and unpredictable. In MARC, the new $m = S[j] + \mathrm{S}[k]$ (is actually $S[i] + S[k]$ compared with n if considering the effect of swap operation) is computed in a similar way as n. The computation of the other two values, $m \hat{} j = (S[i] + S[k]) \hat{} j$ (using the $S[i]$ value before swapping) and $n \hat{} k = (S[i] + S[j]) \hat{} k$, involves all three indices and mixes two different levels of data, i.e., the indices and the state elements located through the indices. These two levels of data are bitwise XORed together. The bitwise XOR operation is linear in $\mathbb{F}_2$, but cannot



(a) ARC4 (output offset = 0)

(b) ARC4 (output offset = 32 bytes)

(c) ARC4 (output offset = 64 bytes)



(d) MARC (output offset = 0)

(e) MARC (output offset = 32 bytes)

(f) MARC (output offset = 64 bytes)

Fig. 1. Avalanche Testing Results for ARC4 KSA and MARC KSA

be handled using pure linear algebra in the residue class ring $\mathbb{Z}/2^n\mathbb{Z}$ or in the field $\mathbb{F}_{2^n}$. In summary, each of the four indices m, n, $m\hat{}j$, and $n\hat{}k$ is either computed similarly as the index n in ARC4 or in a more complicated way.

4 Statistical Testing

In this section we present the statistical testing results of MARC using NIST statistical test suite [16], the new Diehard battery of tests [17], and the more stringent TestU01 batteries of tests [18].

4.1 Testing Results from NIST Statistical Test Suite

The NIST Statistical Test Suite. The NIST tests are based on hypothesis testing. Each test is formulated to test a specific null hypothesis, i.e., a specific sequence of zeroes and ones is random. A probability value (*P-value*) is computed for each test, which summarizes the strength of the evidence against

Table 1. Statistical Testing Results (NIST)

Test	NoP	ARC4		MARC	
		$P\text{-}value_T$	PoS	$P\text{-}value_T$	PoS
1	1	0.161703	991	0.534146	989
2	1	0.146982	998	0.733899	994
3	2	0.572847	989	0.769527	986
		0.928857	989	0.632955	988
4	1	0.229559	984	0.777265	993
5	1	0.595549	991	0.552383	987
6	1	0.134172	992	0.566688	991
7	1	0.274341	988	0.163513	985
8	148	0.470224	989	0.518298	989
9	1	0.404728	992	0.532132	988
10	1	0.029401	988	0.862883	988
11	1	0.676615	990	0.302657	987
12	8	0.299100	$\frac{587}{592}$	0.570090	$\frac{615}{621}$
13	18	0.417622	$\frac{585}{592}$	0.673667	$\frac{614}{621}$
14	2	0.530120	991	0.599693	986
		0.576961	989	0.245490	986
15	1	0.771469	991	0.077607	986

Table 2. Statistical Testing Results (Diehard)

Test	NoP	ARC4		MARC	
		$P\text{-}value_T$	x-p	$P\text{-}value_T$	x-p
1	11	5.3e-10	0	5.4e-006	1
2	2	0.816537	0	0.304126	0
3	33	0.170722	1	0.479073	0
4	5	0.534146	0	0.204439	0
5	2	0.024356	0	0.699313	0
6	26	0.711915	1	0.052326	1
7	20	0.100709	0	0.271619	0
8	82	0.339928	1	0.198214	0
9	1	0.289667	0	0.514124	0
10	25	0.815812	1	0.838472	1
11	11	0.492614	0	0.297118	0
12	11	0.415748	0	0.404407	0
13	21	0.587668	1	0.202783	0
14	1	0.494392	0	0.350485	0
16	3	0.334538	0	0.540878	0
17	4	0.875539	0	0.847183	0

the null hypothesis. It is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non-randomness assessed by the test. If a *P-value* for a test is determined to be equal to 1, then the sequence appears to have perfect randomness. A *P-value* of zero indicates that the sequence appears to be completely non-random. The NIST statistical test suite comprises 15 tests: Frequency, Block Frequency, Cumulative Sums, Runs, Longest Run of Ones, Rank, Discrete Fourier Transform, Non-overlapping Template Matching, Overlapping Template Matching, Universal Statistical, Approximate Entropy, Random Excursions, Random Excursions Variant, Serial, and Linear Complexity.

The empirical results can be interpreted in different ways. The two approaches adopted by NIST are examining the proportion of sequences that pass a statistical test and checking the distribution of *P-values* for uniformity. The first approach is reasonably accurate for large sample sizes (e.g., ≥ 1000 sequences). The second approach needs at least 55 sequences to provide statistically meaningful results.

Testing Setup. We tested and compared the pseudo-random number generation of ARC4 and MARC. For each of them, we tested 1000 pseudo-random sequences, each containing one million bits (125 KB). For each sequence, a random key is generated and used to initialize the generator. This random key can be up to 64 bytes and is generated from a modified ARC4 PRNG using clock value as the input key. The only difference between the modified ARC4 PRNG and the original ARC4 PRNG is that the modified one discards the first 512 bytes of the pseudo-random output.

Testing Results. The testing results are shown in Table 1. The number of *P-values* (NoP) returned by each test is listed in the second column. The $P\text{-}value_T$ measures the uniform distribution of the *P-values* returned by a test and a value equal to or larger than 0.0001 means success. The proportion of sequences (PoS) is given in another column. For 1000 sequences, a minimum value of 980 is required to pass the test. Some tests return more than one *P-value*. If more than 2 *P-values* are returned, we group those *P-values* into two groups based on whether they pass the test or not and give one average value for each group. For a few tests, some sequences may be invalid, in which case we will give out the number of sequences actually used, e.g., $\frac{587}{592}$ means 592 sequences are used and 587 of them pass the test. The testing results show, both generators passed the NIST statistical tests.

4.2 Testing Results from Diehard Battery of Tests

The Diehard Battery of Tests. The Diehard battery of tests are developed by George Marsaglia at Florida State University and first published in 1995. Most of the tests in Diehard return a *P-value*, which should be uniform on [0,1) if the input file contains truly independent random bits. A *P-value* near 0 or 1 indicates deviation from true randomness. This is in contrast with NIST tests, where a bigger *P-value* indicates better randomness. Some of Diehard tests yield more than one *P-value*, in which case a Kolmogorov–Smirnov (KS) test might be run on those *P-values* to produce a single *P-value* that indicates randomness [19]. The new Diehard release contains 17 tests, including some "tough" tests [20]. They are Birthday Spacings (including the new "tough" Birthday Spacings), GCD (new "tough" test), Gorilla (new "tough" test), Overlapping Permutations, Ranks of 31x31 and 32x32 matrices, Ranks of 6x8 Matrices, Monkey Tests on 20-bit Words, Monkey Tests OPSO/OQSO/DNA, Count the 1's in a Stream of Bytes, Count the 1's in Specific Bytes, Parking Lot Test, Minimum Distance Test, Random Spheres Test, The Squeeze Test, Overlapping Sums Test, Runs Up and Down Test, and The Craps Test.

Unlike NIST test suite, Diehard test suite does not provide specific criteria for determining the success or failure of a test, but only says that the *P-values* should be uniform on [0, 1). This results in different interpretations of the testing results. We will evaluate our testing results in two ways, that is, checking the distribution of *P-values* for uniformity and counting the number of *P-values* that are smaller than 0.00001 or larger than 0.9999.

Testing Setup. The C language implementation of Overlapping Sums Test (test 15) is not a faithful interpretation of the author's original Fortran language implementation and none of the (pseudo-)random number generators we tested can pass this test (please see the web page at `http://www.varioustopics.com/cryptography/782655-diehards-overlapping-sum-test.html` for more details). So we exclude this test from our testing. The remaining 16 tests are divided into two groups based on the minimum random sequence size that is needed by each test. GCD, Gorilla, and Overlapping Permutations (test 2, 3, 4) need a much longer random sequence than other tests and are put in a group. All other tests are put in another group. For GCD, Gorilla, and Overlapping Permutations, we tested 50 random sequences for each random number generator, each containing 2176 million bits (272 MB). For other tests, we tested 100 random sequences for each random number generator, each containing 96 million bits (12 MB). Using this setup, at least 100 *P-values* are generated for each test.

Testing Results. The Diehard testing results are given in Table 2. The values given in "x-p" column are numbers of extreme *P-values* (i.e., *P-values* smaller than 0.00001 or larger than 0.9999). Both generators passed all the tests except for the first test. To find the problem, we tested another two (pseudo-)random number generators: SHA1 (running in counter mode) and QRNG (a quantum random number generator available at `http://qrng.physik.hu-berlin.de/`). Both of them failed this test too, which suggests it could be a problem with the test itself. ARC4 has a few more extreme *P-values* than MARC, but is still within the normal range (one out of 10^5).

4.3 Testing Results from TestU01 Batteries of Tests

The TestU01 Batteries of Tests. TestU01 is the most comprehensive statistical test suite that is publically available so far. It is a software library implemented in the ANSI C language. It offers a collection of utilities for the empirical statistical testing of uniform random number generators. Six pre-defined batteries of tests are available in TestU01. They are SmallCrush, Crush, BigCrush, Rabbit, Alphabit, and BlockAlphabit. Any (pseudo-)random number generator that implements both the *double (*GetU01) (void *param, void *state)* and *unsigned long (*GetBits) (void *param, void *state)* interfaces defined in TestU01 can use all the 6 pre-defined batteries of tests.

In the current version, Crush uses approximately 2^{35} random numbers and applies 96 statistical tests (it computes a total of 144 test statistics and *P-values*), whereas BigCrush uses approximately 2^{38} random numbers and applies 106 tests (it computes 160 test statistics and *P-values*). When invoking the battery Rabbit, Alphabit, and BlockAlphabit, one must specify the number of bits available for each test. Other parameters of each test are chosen automatically as a function of the number of available bits. Rabbit and Alphabit apply 40 and 17 different statistical tests respectively. BlockAlphabit applies the Alphabit battery of tests repeatedly to a generator or a binary file after reordering the bits by blocks of different sizes (with sizes of 2, 4, 8, 16, 32 bits).

Table 3. Statistical Testing Results (TestU01)

Battery	Parameters	Tests	NoP	Failures	
				ARC4	MARC
SmallCrush	Built-in	10	15	0	0
Crush	Built-in	96	144	0	0
BigCrush	Built-in	106	160	0	0
Rabbit	32×10^9 bits	26	40	0	0
Alphabit	32×10^9 bits	9	17	0	0
BlockAlphabit	32×10^9 bits	6×9	102	0	0

Testing Setup. TestU01 requires much more (pseudo-)random numbers than the NIST and Diehard suites. It takes more than 12 hours to run all the 6 batteries on our machine. Each generator is implemented in C programming language and created as a *unif01_Gen* object. Both *GetU01()* and *GetBits()* interfaces are implemented for each generator so that all 6 batteries of tests can be applied. Built-in parameters are used for SmallCrush, Crush, and BigCrush. For Rabbit, Alphabit, and BlockAlphabit, the size of bit sequence is set to 32×10^9.

Testing Results. The testing results from TestU01 batteries of tests are given in Table 3. The 6 batteries of tests return 478 *P-values*. Each *P-value* is interpreted as follows: a value in the interval $[10^{-3}, 1 - 10^{-3}]$ means the test is successful; a value outside the interval $[10^{-10}, 1 - 10^{-10}]$ (i.e., too close to 0 or 1) indicates a clear failure; and a value in between is a suspect value. For a suspect *P-value*, we repeat the test 5 times. If no failures or suspect *P-values* are observed during the retries, we clear the test; otherwise we mark the test as failed. Both ARC4 and MARC passed all the tests, but ARC4 has 2 suspect *P-values* during the first run while MARC has none.

5 Performance Testing

The speed testing results for MARC are given in Table 4. The testing results for ARC4 and the four software-efficient finalists of eStream are also included in the table for comparison. The testing is done for a software implementation using C programming language. The C implementation closely follows the pseudo code given in section 2. There are no special optimizations done at the source code level except that register variables are used to minimize memory access whenever possible. Most modern compilers are smart enough and know more about code generation than the developer [21]. Therefore we leave optimizations largely to the compiler.

Table 4. Pseudo-Random Number Generation Speed (cycle/byte)

Generator	Sequence size (KB)					
	1	5	10	100	1000	10000
ARC4	9.53	7.67	7.09	6.98	7.04	7.04
MARC	17.46	6.60	5.21	3.98	3.89	3.86
HC-128	55.21	13.27	7.96	3.58	3.15	3.11
Rabbit	12.20	10.06	9.63	9.51	9.52	9.49
Salsa20	8.94	8.95	8.95	8.89	8.90	8.88
Sosemanuk	48.67	13.48	9.70	5.79	5.61	5.36

Both 32-bit and 64-bit executables compiled using Microsoft Visual C/C++ Optimizing Compiler Version 16 with option /O2 (optimized for maximum speed) are tested. ARC4 and eStream finalists are not 64-bit algorithms, but their 64-bit executables seem running slightly faster on our machine (Intel Core i3 370M, 2.4GHz, 64 KB L1 data cache, 64 KB L1 instruction cache, 512 KB L2 cache) and are chosen for the testing.

For each sequence size, we run each executable 30 times and get the average value of the top 3 speeds. The reason we exclude low speeds in our calculation is that the measured cycles may contain contributions from some system processes that we cannot stop and the small cycles more likely reflect the actual performance. MARC outperforms all generators except HC-128.

6 Conclusion

In this paper we have presented a variant of ARC4 called MARC. It enhances the key scheduling algorithm of ARC4 and resists to known attacks that exploit the weakness in ARC4's key scheduling. It also improves the pseudo-random generation algorithm of ARC4 and has a better performance than ARC4 and most eStream finalists. It passed all the NIST statistical tests, the new Diehard battery of tests, and all the 6 TestU01 batteries of tests.

References

1. Schneier, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn. John Wiley & Sons (1995)
2. Fluhrer, S., Mantin, I., Shamir, A.: Weaknesses in the Key Scheduling Algorithm of RC4. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 1–24. Springer, Heidelberg (2001)
3. RSA Security Response to Weaknesses in Key Scheduling Algorithm of RC4, `http://www.rsa.com/rsalabs/node.asp?id=2009`

4. Wu, H.: The Stream Cipher HC-128, http://www.ecrypt.eu.org/stream/hcpf.html
5. Boesgaard, M., et al.: The Stream Cipher Rabbit. eSTREAM report 2005/024 (2005), http://www.ecrypt.eu.org/stream/papers.html
6. Bernstein, D.J.: Salsa20/8 and Salsa20/12. eSTREAM report 2006/007 (2006), http://www.ecrypt.eu.org/stream/papers.html
7. Berbain, C., et al.: Sosemanuk, a fast software-oriented stream cipher. eSTREAM report 2005/027 (2005), http://www.ecrypt.eu.org/stream/papers.html
8. Paul, S., Preneel, B.: A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 245–259. Springer, Heidelberg (2004)
9. Zoltak, B.: VMPC One-Way Function and Stream Cipher. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 210–225. Springer, Heidelberg (2004)
10. Maitra, S., Paul, G.: Analysis of RC4 and Proposal of Additional Layers for Better Security Margin. In: International Conference on Cryptology in India. IIT, Kharagpur (2008)
11. Khine, L.L.: A New Variant of RC4 Stream Cipher. World Academy of Science, Engineering and Technology 50 (2009)
12. Roos, A.: A Class of Weak Keys in the RC4 Stream Cipher. Posting to sci.crypt (1995)
13. Golić, J.D.: Linear Statistical Weakness of Alleged RC4 Keystream Generator. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 226–238. Springer, Heidelberg (1997)
14. Klein, A.: Attacks on the RC4 Stream Cipher. Designs, Codes and Cryptography 48, 269–286 (2008)
15. Tews, E., Weinmann, R.-P., Pyshkin, A.: Breaking 104 Bit WEP in Less Than 60 Seconds. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 188–202. Springer, Heidelberg (2008)
16. Runkin, A., et al.: Statistical Test Suite for Random and Pseudo Random Number Generators for Cryptographic Applications. NIST special publication 800-22
17. Marsaglia, G.: DIEHARD Battery of Tests. New version, http://www.csis.hku.hk/~diehard/
18. L'Ecuyer, P., Simard, R.J.: Testu01: A C Library for Empirical Testing of Random Number Generators. ACM Trans. Math. Softw. 33(4) (2007)
19. Soong, T.T.: Fundamentals of Probability and Statistics for Engineers, p. 327. John-Wiley and Sons Ltd. (2004) ISBN: 0470868147
20. Marsaglia, G., Tsang, W.: Some Difficult-to-Pass Tests of Randomness. Journal Statistical Software 7(3) (2002)
21. Leitner, F.V.: Source Code Optimization, http://www.linux-kongress.org/2009/slides/compiler_survey_felix_von_leitner.pdf

Detection of HTTP-GET Attack with Clustering and Information Theoretic Measurements

Pawel Chwalinski, Roman Belavkin, and Xiaochun Cheng

School of Engineering and Information Sciences, Middlesex University, London, UK
{p.chwalinski,r.belavkin,x.cheng}@mdx.ac.uk

Abstract. One of the attacks observed against HTTP protocol is HTTP-GET attack using sequences of requests to limit accessibility of webservers. This attack has been researched in this report, and a novel, off-line clustering technique has been developed to tackle it. In general, the technique uses entropy-based clustering and application of information theoretical measurements to distinguish among legitimate and attacking sequences.

It has been presented that the introduced method allows for formation of recent patterns of behaviours observed at a webserver, that remain unknown for the attackers. Subsequently, statistical and information theoretical metrics are introduced to measure difference between a sequence of requests, and legitimate patterns of behaviour.The method recognises more than 80% of legitimate and attacking sequences, regardless of strategies chosen by attackers.

Keywords: HTTP-GET Attack, Information Theory, Clustering, Intrusion Detection.

1 Introduction

In theory, web-browsing is based on GET requests generated by sender. For two-node communication, sender repeats sending GET requests to receiver; who in return replies with requested data.

Scenario for running the HTTP-GET attack is alike with standard techniques used by attackers performing Distributed Denial of Service (DDoS) attack. At first, a number of hosts are infected with a computer virus, and attackers take control over the number of *zombies* (i.e. infected machines). Subsequently, attackers generate a large volume of traffic at victim's website with the application of infected hosts, which continue to send GET requests.

The problem that is tackled in this paper is not related to virus detection or prevention; nor is it related to malicious code propagation. On the contrary, an off-line detection of flooding hosts has been investigated. The difficulty in detection of the attack stems from legitimate nature of attacking hosts. Rather than sending ill-formatted network packets, attackers make their zombies comply with computer network regulations, and mimic genuine activity. In addition, *low rate* arrival of zombies and their request frequency, make them look as system-friendly connections for rate-based Intrusion Detection Systems (IDS).

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 45–61, 2013.

Indeed, some researchers claim that the *intention* is the only difference among connections [26]. In addition, because of stealthy appearance of the zombies, their activity becomes indistinguishable from legitimate data flow [20]. Moreover, detection mechanisms based on traffic characteristics become invalid [27]. Yet another obstacle with HTTP-GET attack is a ubiquitous problem in intrusion detection called *flash crowd* effect. It is a situation when many legitimate users visit a website in a short time. Thus, the webserver experiences the influx of new hosts, but it should be recognised as lawful activity.

Therefore, in general, the detection problem comes down to *intention classification*, and in fact this is the main problem of this research. One approach to differentiating between valid and attacking connections is to focus on the recent behaviour of actual users. Specifically, given the actual structure of a website, legitimate connections visit similar pages in groups. As a result, sessions interested in similar categories, avoid browsing other categories that again seem to be interesting to different groups of users. Thus, because the attackers do not know the actual interest in a website's content, they will fail to reproduce actual sequences of requests.

Therefore, the main contribution of this work is the development of new clustering algorithm that allows for recent interest discovery among visiting users. The clustering approach has been inspired by work of Barbará [3], where entropy based clustering is used to cluster categorical data points. In this work, this approach has been applied to *sequences* of categorical requests. In addition, three techniques have been introduced to improve the algorithm speed, and undesirable effect of sequences order (see Sect. 2.3).

The organisation of the paper is as follows. Initially, the description of the algorithm is presented with techniques that improve its performance. Subsequently, its is shown how to assign training and attacking sequences to existing clusters. Next, two strategies of attacking hosts are presented that appear to closely mimic human agents. However, it is shown that it is difficult to repeat behaviour encoded with the clustering algorithm. Subsequently, statistical measurements to detect attacking hosts are introduced; and detection performance of the measurements is provided. The paper concludes with the related work section, and description of future work.

2 Clustering

In order to discover recent interest in a website's content, the clustering algorithm is introduced to group sequences of requests observed at a server.

2.1 Definitions and Notations

Suppose there is a dataset D containing n sequences of requests $\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_n}$. Subsequently, the dataset is randomly divided into two sets: D^T for training (i.e. clustering), and D^V for validating; each containing n^T and n^V sequences respectively. Set D^T is composed of roughly three times as many sequences as

D^V. Moreover, a sample space Ω is introduced, containing n^C numerical labels, representing actual categories of a webserver, such that $\Omega=\{1,2,\ldots,n^C\}$.

In addition, set D^A (explained in detail in Sect. 3) is generated consisting of $n^A=n^V$ attacking sequences. Furthermore, $D^{V\bigcup A}=D^V\bigcup D^A$, containing $n^{V\bigcup A}=n^V+n^A$ sequences will be used for detection of attacking sequences. For each $i, 1\leq i\leq n$, $\mathbf{s_i}$ is a realization of a random sequence that takes on values in Ω. Moreover, each sequence $\mathbf{s_i}$ consists of $n_i \in \{r_\bigwedge,\ldots,r_\bigvee\}$ numbers such that $\mathbf{s_i}=(s_{i,1}, s_{i,2},\ldots,s_{i,j},\ldots,s_{i,n_i})$, $r_\bigwedge, r_\bigvee$ denote minimum and maximum number of requests respectively, and each $s_{i,j}$ takes on values in Ω.

In general, for categorical data points entropy-based clustering is used [3],[16]. For a joint distribution $p(x,y)$, entropy $h(x,y)$ is calculated as [18]:

$$h(x,y)=-\sum_x\sum_y p(x,y)\log p(x,y) \tag{1}$$

Generally, high entropy corresponds to flatter distribution, whereas smaller entropy denotes skewer distribution. Thus, for IDS purposes one seeks clusters with smaller entropy to group elements representing similar behaviour [15]. Random variables x and y used in joint distribution in (1) correspond to observing a pair of requests $(x,y)\in\Omega\times\Omega$.

2.2 Clustering Criteria

In order to group sessions stored in D^T into k clusters, k *empirical* joint distributions $p(x,y)$ are obtained by analysing sequences from each cluster. Specifically, if there is set $\mathbf{C}=\{C_1,C_2,\ldots,C_k\}$ of k clusters, the goal is to minimise the average entropy of the clusters, which is computed as follows:

$$\mathbb{E}\{\mathbf{H}\}=\sum_{j=1}^{k}\left(\frac{|C_j|}{|D^T|}\{h_j\}\right) \tag{2}$$

Therefore, having grouped $b, 1\leq b\leq n^T$ sequences from D^T given (2), every time a sequence $\mathbf{s_i}, b<i\leq n^T$ is to be added to the composition of clusters $\mathbf{C}$, a cluster C_j, $1\leq j\leq k$ is picked where adding sequence $\mathbf{s_i}$ decreases the *scaled* (i.e. multiplied by $\frac{|C_j|}{|D^T|}$) cluster's entropy the most. Sometimes, creating a *new* cluster is actually the best placement for sequence $\mathbf{s_i}$. Therefore, k is not fixed; it changes while adding new sequences to $\mathbf{C}$, and depends on the factors described below.

2.3 Minimising the Effect of Sequences Order

It is important to note that the order of sequences can play a crucial role in the clustering algorithm (i.e. decreasing or increasing its performance). In order to attenuate the unwanted effect of ordering, the following potentially improving processes have been introduced: Re-clustering, Merging and Partitioning.

Re-clustering. Re-clustering process is proposed to find a better cluster for a sequence $\mathbf{s_i}, 1{\leq}i{\leq}b$, including clusters that were not present during initial addition of $\mathbf{s_i}$, after having processed $b, 1{\leq}b{\leq}n^T$ sequences from D^T. In other words, suppose that while grouping $\mathbf{s_i}$ there are k clusters, and $\mathbf{s_i}$ has been added to $C_j, 1{\leq}j{\leq}k$. It might be the case, that after having grouped other b sequences, there exists another cluster $C_{j'}, 1{\leq}j'{\leq}k, j' \neq j$ such that placing the previously added $\mathbf{s_i}$ inside $C_{j'}$, minimises (2) further. Therefore, after processing a batch of b sequences, the algorithm is stopped, and the whole set $\mathbf{C}$ is re-clustered.

Merging. Yet another process that can minimise (2) is merging. While connections are being added to cluster composition $\mathbf{C}$, it might be the case that there are some clusters that pose similar characteristics, however they exist as two different instances.

One way of comparing two joint distributions $p_i(x, y), 1{\leq}i{\leq}k$, and $p_j(x, y)$, $1{\leq}j{\leq}k$ is the application of Kullback-Leibler (KL) divergence [10] formula:

$$D_{KL}(p_i||p_j){=}\sum_{x,y\in\Omega}\sum p_i(x,y)\log\frac{p_i(x,y)}{p_j(x,y)} \tag{3}$$

However, KL divergence is not symmetric, nor does it satisfy triangle inequality. Thus, while deciding whether or not two clusters C_i and C_j are similar, a dual KL divergence is calculated given the corresponding $p_i(x, y)$ and $p_j(x, y)$. Then, empirically a threshold η_{KL} is calculated specifying whether two C_i and C_j are similar. Thus, C_i and C_j are merged when $D_{KL}(p_i||p_j){\leq}\eta_{KL}$, and $D_{KL}(p_j||p_i){\leq}\eta_{KL}$.

Partitioning. The clustering approach based on minimisation of (2) is *computationally expensive* [3]. In order to increase the speed of the algorithm, a partitioning of connections is introduced.

Before the algorithm is applied, dataset D^T is partitioned into a set of partitions $\mathbf{B}$ containing a small number of partitions p. Each $B_i \in \mathbf{B}, 1{\leq}i{\leq}p$ is a *range* of the ordered sequences' indexes, such that $\mathbf{B_i}{=}\{B_{i,1}, \ldots B_{i,d_i}\}$, where $d_i, 1{\leq}d_i{\leq}n^T$ denotes the index of a d_i-th sequence from D^T corresponding to the last element's index from $\mathbf{B_i}$. For two adjacent ranges $\mathbf{B_i}$, and $\mathbf{B_j}{=}\{B_{j,1}, \ldots, B_{j,d_j}\}, 1{\leq}j{\leq}p, j \neq i$, the following relationship exists: $B_{i,1}{<}B_{i,d_i}{<}B_{j,1}{<}B_{j,d_j} \leq n^T$. For each partition $\mathbf{B_i}$, the clustering algorithm is applied with re-clustering and merging to prepare k_i clusters $\mathbf{C_i^B}{=}\{C_{i,1}, C_{i,2}, \ldots, C_{i,k_i}\}$ for block $\mathbf{B_i}$. When each partition of indexes has been processed, each set of clusters $\mathbf{C_i^B}$ contains best arrangements of k_i clusters, given (2), for its own sample of connections *only*. Afterwards, the sets of clusters $\mathbf{C_i^B}$ are merged together. Subsequently, merging and re-clustering is performed on the newly obtained set of clusters $\mathbf{C}{=}\{\mathbf{C_1^B}, \mathbf{C_2^B}, \ldots, \mathbf{C_p^B}\}$ until there is no improvement of (2). This process saves time, because merging and re-clustering are not as computationally expensive as minimising (2) for a large sample of connections, allowing k to dynamically change.

2.4 Algorithm - The Summary

In general, the clustering algorithm can be represented with Algorithm 1. Initially, connections are divided into p same-length blocks. For each block, the clustering routine is performed to minimise its own version of (2). Merging and re-clustering are performed as well after every b connections have been processed within each $\mathbf{B_i}, 1 \leq i \leq p$ block. When clustering of each block $\mathbf{B_i}$ has been finished and corresponding sets of clusters have been prepared, there are many clusters spread across composition $\mathbf{B}$. Thus, the clusters from each $\mathbf{B_i}$ are gathered and analysed as one set of clusters $\mathbf{C}$. Subsequently, until the value of (2) cannot be minimised further, set $\mathbf{C}$ is merged and re-clustered alternately.

Algorithm 1. Clustering algorithm

```
1: divide connections into p partitions B
2: for all B_i ∈ B, 1≤i≤p do
3:     j = B_{i,1}
4:     for all s_j, j ∈ B_i do
5:         no_of_processed_connections=0
6:         while no_of_processed_connections<b and j ≤ B_{i,d_i} do
7:             if adding s_j to C_{i,t}, 1≤t≤k_i minimises (2) then
8:                 assign s_j to C_{i,t}
9:             else
10:                Create new cluster C_{n_k} containing s_j
                   n_k=k_i + 1, k_i=n_k
11:            end if
12:            no_of_processed_connections + +, j + +
13:        end while
14:        Re-cluster B_i, Merge B_i
15:    end for
16: end for
17: integrate all C_i ∈ B to get C
18: repeat
19:    Re-cluster C,Merge C
20: until (2) cannot be minimised further
```

2.5 Results of the Clustering Algorithm

To present the algorithm performance, a dataset has been obtained from a day-long activity recorded at `http://msnbc.com`, on the 28^{th} of September, 1999. This dataset is composed of sequences of numbers, corresponding to categories that were visited by users. The dataset has been used before in detection of web interest [7],[19] and clustering of web sessions [11].

For the obtained dataset, sequences containing at least $r_{\bigwedge}$=2 and maximum $r_{\bigvee}$=30 requests, and those that have visited at least 2 categories are analysed. Thus, there are n^T=296,071 training and n^V=82,518 sequences; and there are n^C=17 categories.

Initially, (i.e. when k=1) the entropy of cluster composition $\mathbf{C}=\{C_1\}$ amounts to h_k=5.9926. Having applied the clustering algorithm with b=1000, η_{KL}=1.5, k=1078 clusters have been generated. As a result, the ordered entropy range is introduced $\mathbf{H}=\{h_1, h_2, \ldots, h_k\}$, such that $h_1 \equiv \min \mathbf{H}$=1.2789 is the entropy of a cluster having the smallest entropy h_1. Similarly, $h_k \equiv \max \mathbf{H}$=7.9297 is the maximum entropy after the application of the algorithm. The average entropy of the composition is $\hat{\mathbf{H}}=\frac{\sum_{i=1}^{k} h_k}{k}$=4.3815, and the scaled average entropy amounts to $\mathbb{E}\{\mathbf{H}\}$=3.5529. Table 1 contains information about required time

Table 1. Clustering time depending on different values of n^T

n^T	50,000	100,000	150,000	200,000	250,000	296,071
Time	54 min	1 hr 33 min	3 hr 5 min	5 hr 59 min	7 hr 22 min	8 hr 48 min

to cluster batches of various sizes. The clustering experiments were run on a stand-alone machine containing Intel Core Duo CPU 3.32 GHz, with Matlab R2012a installed. Note well, that Matlab could have worsened the result because of its high-level programming-language dependency. In addition, the time periods are consistent with [3], and should allow to learn daily activity of web users overnight.

2.6 Cluster Assignment Criterion

Having clustered the connections form D^T, the connections from D^V and D^A should be assigned to the set of clusters $\mathbf{C}$. The actual detection technique of attacking hosts is provided in Sect. 5.

In order to find a best cluster for a sequence $\mathbf{s_i}, 1 \leq i \leq n^{V \cup A}$, a technique based on maximum likelihood principle is introduced. This method is a common practice, while looking for a best cluster for new connection $\mathbf{s_i}$ [16]. In simple terms, the best model (i.e. cluster) is the one, under which an observation attains *the highest likelihood*. Probability of observing sequence $\mathbf{s_i}$ inside cluster C_j, $1 \leq j \leq k$ with corresponding joint distribution $p_j(x, y)$ can be thought of in the following way (having assumed that two consecutive requests $s_{i,l}, s_{i,l+1}, 1 \leq l < n_i$ are independent):

$$p_j(s_{i,1}, s_{i,2}, \ldots, s_{i,n_i}) = p_j(s_{i,1}) \prod_{l=2}^{n_i} p_j(s_{i,l} | s_{i,l-1}) \tag{4}$$

Subsequently, the probability of generating sequence $\mathbf{s_i}$ is calculated for each cluster, and then C_j is chosen for which (5) attains the highest value:

$$C_j = \underset{C_1, C_2, \ldots, C_k}{\arg\max} \log \mathcal{L}(\mathbf{s_i}; \mathbf{C}) = P(\mathbf{s_i} | \mathbf{C}) \tag{5}$$

where $\mathcal{L}(\mathbf{s_i}; \mathbf{C})$ is the likelihood of obtaining connection $\mathbf{s_i}$ inside C_j, and $P(\mathbf{s_i}|\mathbf{C})$ is likelihood of the sequence given a collection of all joint probability distributions generated by each cluster from $\mathbf{C}$.

3 Strategies of Attacking Hosts

For many years, low likelihood of any activity has been considered anomalous [5]. As a result, D^A could be composed of sequences whose requests are uniformly distributed. However, it is crucial to define realistically the attacking hosts' strategies. Therefore, we have decided to define attacking strategies that remind human behaviour as follows.

Suppose that having requested a link from category $c \in \Omega$, a programmed zombie faces a decision-making problem whether with probability p^R to *remain* inside the same category (i.e. request a link from c-th category again), or with probability $p^M{=}1 - p^R$ *move* to any of the remaining categories (i.e. request a link from category $c_m \in \Omega \backslash \{c\}$). Given this decision-making problem, two attacking strategies are considered: frequently-changing and rarely-changing.

3.1 Frequently-Changing Hosts

Frequently-changing hosts are the ones that tend to change categories more frequently comparing to the rarely-changing zombies. Specifically, for this type of attacking hosts $p^M{=}p^R{=}0.5$. In other words, while making requests zombies will stay inside or move to another category with equal probability $p^M{=}p^R{=}0.5$. Observe that a frequently-changing host changes categories rarely than a uniformly programmed host mentioned above, and thus its behaviour reminds human activity more.

A detection of these hosts should focus on rarely observed transitions. However, there are high-entropy clusters inside which, relatively more unique transitions are observed comparing to low-entropy clusters. For this reason attackers find it easier to fit in inside high-entropy clusters. However, it remains unknown for attackers how many times each category is requested on average inside these clusters. Therefore, Mahalanobis distance has been chosen to detect sessions deviating from the pattern of the average categorical interest. In addition, mutual information is applied to measure likelihood of transition between sessions. Both techniques are described in the following section.

3.2 Rarely-Changing Hosts

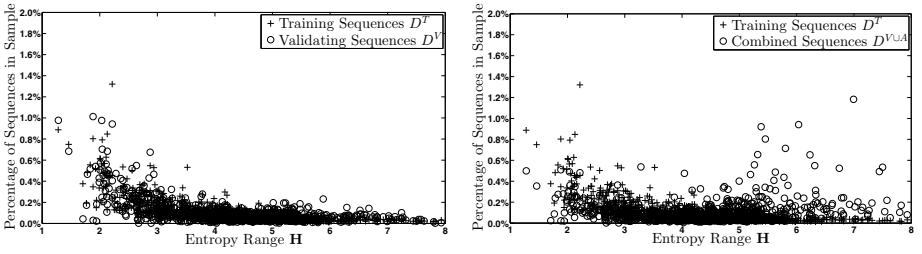
Suppose that one wants to generate sequences D^A similar to D^T. Because information encoded in $\mathbf{C}$ remains for attackers unknown, suppose they have found out how many different categories are visited on average inside D^T. As a result, while analysing D^T, a vector $\mathbf{e^T}$ has been calculated containing the expected number of various categories, given the number of requests, such that $\mathbf{e^T}{=}[\mathbb{E}\{n^c_{r_\bigwedge}\}, \mathbb{E}\{n^c_{r_{\bigwedge+1}}\}, \ldots, \mathbb{E}\{n^c_{r_{\bigwedge+t}}\}, \ldots, \mathbb{E}\{n^c_{r_\bigvee}\}]$, where $\mathbb{E}\{n^c_{r_{\bigwedge+t}}\}$ denotes the expected number of visited categories after $\bigwedge{+}t$ requests. Subsequently, $p^R{=}0.92, p^M{=}0.08$ have been estimated to create randomly generated dataset D^A, whose corresponding $\mathbf{e^A}$ is similar to $\mathbf{e^T}$. Therefore, these zombies will *remain* for many more requests in one category than the above-mentioned counterparts.

Clearly, the values of p^R and p^M suggest that human visitors prefer to request only few categories per visit; and it is relatively easy to come up with this idea for attackers. However, because attackers aim is to flood webservers and make their zombies invisible for IDS, they program their zombies to remain inside a category for longer. Note well that Mahalanobis distance will not be as effective as in the case of the frequently-changing hosts. It stems from the fact that rarely-changing hosts generate sequences more similar to patters encoded inside **C**. As a result, one could be interested in the likelihood of the longest same-request segment introduced in Sect. 5.3. This will be supported with mutual information comparisons for clusters where the increased number of attacking connections is suspected.

The introduced statistical measurements are described in Sect. 5. What follows, is a presentation of session distribution across clusters.

4 Detection of Attacking Attempt

It is assumed that, if there is a set of new links or files uploaded on-line that propel, say, "behaviour" $C_i, 1 \leq i \leq k$, then one could expect a surge of connections for C_i within a limited time period. Similar assumptions and observations have been reported in scientific literature [8],[27]. The same regularity is observed while comparing D^V (circles) against D^T (crosses) (see Fig. 1a or 2a). Specifically, most sequences from D^T and D^V fall inside low-entropy clusters. When

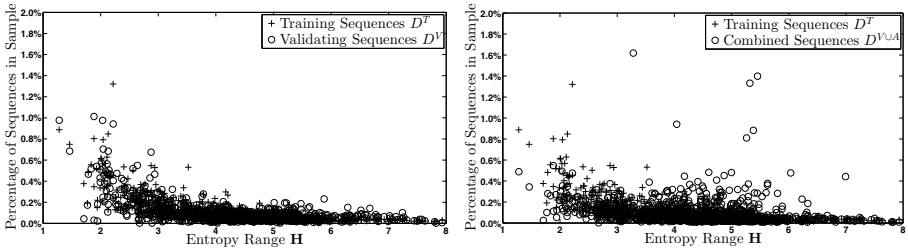


(a) Percentage of sequences from datasets D^T and D^V

(b) Percentage of sequences from datasets D^T and $D^{V \bigcup A}$

Fig. 1. Comparison of sequence distribution with (1b), and without (1a) attacking sequences. Evidently, the distribution changes with the arrival of the frequently-changing sequences.

the set $D^{V \bigcup A}$ consists of sequences generated by the frequently-changing hosts, one can learn that the distribution of $D^{V \bigcup A}$ has switched to the right hand side of the chart (see Fig. 1b). It stems from the fact that the frequently-changing hosts will populate clusters with higher entropies (i.e. clusters where relatively more categories are visited).



(a) Percentage of sequences from sets D^T and D^V

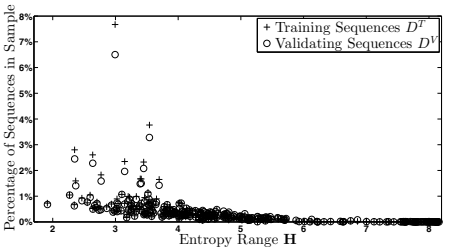
(b) Percentage of sequences from sets D^T and $D^{V \cup A}$

Fig. 2. Comparison of sequence distribution with (2b), and without (2a) attacking sequences. The distribution of the set $D^{V \cup A}$ with rarely-changing hosts shifts towards the expected distribution (Fig. 2a), but still differs.

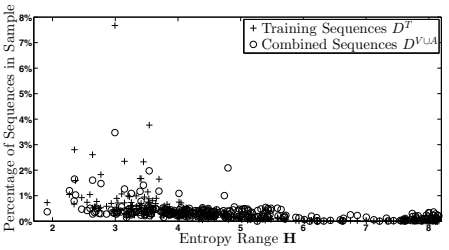
Similarly, when the attacking sequences inside set $D^{V \cup A}$ follow pattern of the rarely-changing hosts defined in Sect. 3.2, the distribution has shifted from the right-hand side of the chart, but still deviates from the original distribution presented in Fig. 2a. The comparison of sequence distributions can give a hint which type of attack has been observed. Thus, by analysing the distributions, one can evaluate whether legitimate (Fig. 1a, Fig. 2a), the frequently-changing (Fig. 1b) or the rarely-changing (Fig. 2b) connections have been observed at the system.

The introduced solution has been compared against K-means algorithm. The cosine similarity distance has been used as it captures best human interest in websites [23]. Recall that the pairs of requests $(x, y) \in \Omega \times \Omega$ are used in entropy minimisation, and in total there are $(n^C)^2$ possible pairs. Therefore, connections from D^T and $D^{V \cup A}$ are transformed into $(n^C)^2$ long vectors, where each component contains a counter of how many times each pair has been requested, and $K = (n^C)^2$.

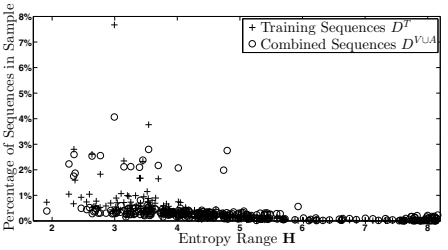
Subsequently, an entropy range **H** is calculated for K generated clusters, and clusters are sorted in ascending order of entropy values. Initially, the distributions of connections from D^T, and D^V have been compared against each other. Again, sequences from both sets tend to populate low-entropy clusters (see Fig. 3a). However, very similar distributions of sequences are observed when $D^{V \cup A}$ is composed of validating and frequently-changing (Fig. 3b) or validating and rarely-changing (Fig. 3c) sequences. This shows, that K-means clustering does not provide as much insight into attackers strategy as the entropy-based clustering. In addition, while comparing **H** ranges (for the entropy-based and K-means algorithms), one can note that entropy-based clustering produces clusters with lower entropy, thus provides better arrangement of data. Moreover, k has been determined by the corresponding algorithm's calculations and is not manually set as K.



(a) Training D^T and validating D^V sequences



(b) Training D^T with validating and frequently-changing $D^{V \cup A}$ sequences



(c) Training D^T with validating and rarely-changing $D^{V \cup A}$ sequences

Fig. 3. Distribution of sessions with K-means clustering. Despite using different sets (i.e. validating sequences (3a), validating and frequently-changing sequences (3b) or validating and rarely-changing sequences (3c)) the distributions are similar and do not give any insights into the nature of processing batch .

5 Detection of Attacking Hosts

It might be the case that the attacking hosts guess the actual pattern of behaviour. However, because internet attackers cannot have access to the profile generated by legitimate users, the attackers cannot know how many times and in which order the correct categories have been requested. Therefore, it is crucial to develop a measure to check how "anomalous" one sequence behaves.

5.1 Mahalanobis Distance

To measure expected categorical interest among connections within a cluster, one can use Mahalanobis distance. For this purpose, for each cluster $C_i, 1 \leq i \leq k$ a corresponding covariance matrix Σ_i is calculated, together with a vector of average categorical requests $\mu_i = [\mu_{i,1}, \mu_{i,2}, \ldots, \mu_{i,n^C}]$. Subsequently, each *training* connection $\mathbf{s_j}, 1 \leq j \leq |C_i|$ from C_i, is transformed into a vector form $\mathbf{v}_j = [v_{j,1}, v_{j,2}, \ldots, v_{j,l}, \ldots, v_{j,n^C}]$, where each $v_{j,l}$ denotes how many times l-th category has been requested during session $\mathbf{s_j}$. As a result, *training* Mahalanobis distance can be calculated in the following way:

$$d_M(\mathbf{v_j}, C_i) = \sqrt{(\mathbf{v_j} - \mu_\mathbf{i})^T \Sigma_i^{-1} (\mathbf{v_j} - \mu_\mathbf{i})} \tag{6}$$

Subsequently, vectors of Mahalanobis distances $\mathbf{m_i^M}=[m_{i,\bigwedge}^M, m_{i,\bigwedge+1}^M, \ldots, m_{i,t}^M, \ldots, m_{i,r_\bigvee}^M]$ are obtained, where each t-th component contains the maximum Mahalanobis value observed inside C_i for t-request-long sequences. For any t component for which $m_{i,t}^M=0$, maximum value of Mahalanobis distance $\eta_i^{M_\bigvee}$ will be stored, such that $\eta_i^{M_\bigvee} \equiv \max_{v_1,v_2,\ldots,v_{|C_i|}} d_M(\cdot, C_i)$. The reason why $\eta_i^{M_\bigvee}$ is introduced, stems from the fact that sometimes there is a cluster $C_l, 1\leq l\leq k$, where sequences containing t requests are not observed. To observe sequences containing t requests inside C_l is assumed to be anomalous. However, because it is preferred to let "well-fitted" connections through (instead of setting $m_{l,t}^M=0$, and effectively blocking deviating t-request-long sequences), therefore the maximum values are set (keeping in mind the possibility of allowing more legitimate connections in).

In principle, if a sequence $\mathbf{s_j}, 1\leq j\leq n^{V\bigcup A}$ has been assigned to cluster C_i its Mahalanobis distance $d_M(\mathbf{s_j}, C_i)$ will be computed. As a result, if $d_M(\mathbf{s_j}, C_i)\leq \mathbf{m_i^M}[n^j]$, then $\mathbf{s_j}$ will be marked as legitimate connections. Otherwise, $\mathbf{s_j}$ will be marked as anomalous.

5.2 Mutual Information and Statistical Independence as Anomalous Measurement

Mutual information between two discrete random variables X, Y is defined in the following way:

$$I(X;Y)=\sum_{y\in Y, x\in X} p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

Mutual information provides a measure of independence between two random variables. It is equal to zero (i.e. $I(X;Y)=0$) for independent variables, and increases when knowing realisation of one variable reduces uncertainty of the other. As a result, a measure is introduced based on statistical independence.

Suppose that for each cluster $C_i, 1\leq i\leq k$, $\frac{n^C(n^C-1)}{2}-n^c$ joint probability distributions $p(x,y)$ are obtained, between two different categories $x, y \in \Omega$ *only* (i.e. where $x\neq y$). In simpler words, if one considers set $\Omega\times\Omega$ for each cluster C_i, then $\frac{n^C(n^C-1)}{2}-n^c$ joint distributions are obtained between two *different* categories (i.e. distributions from either *strictly* upper or lower triangular part from $\Omega\times\Omega$ matrix are considered).

Subsequently, for each cluster C_i a two-dimensional vector $\mathbf{m}_i$ is introduced, where each j-th column contains $\mathbf{m}_{i,j}^{\bigwedge}$ minimum and maximum $\mathbf{m}_{i,j}^{\bigvee}$ value of mutual information while requesting j-th category.

As a result, while analysing sequences from D^T, mutual information is calculated between pairs of two consecutive and *different* categorical requests. In other words, for sequence $\mathbf{s_l}=(s_{l,1}, s_{l,2}, \ldots, s_{l,t}, \ldots, s_{l,n_l}), 1\leq l\leq n^T$ if the following holds $s_{l,t-1}\neq s_{l,t}\neq s_{l,t+1}\neq s_{l,t-1}$, then two dependencies will be calculated: $I(s_{l,t-1}; s_{l,t})$, and $I(s_{l,t}; s_{l,t+1})$. The values of $I(s_{l,t-1}; s_{l,t})$, and $I(s_{l,t}; s_{l,t+1})$ are stored in vector $\mathfrak{m}_l$ containing $|\mathfrak{m}_l|$ elements.

Because it is assumed that human users will follow similar patterns of behaviour in D^V as in D^T, then it should not be the case that two different categorical requests from sequence $\mathbf{s_l}$, say $s_{l,t-1}, s_{l,t}$, produce value of mutual information greater or smaller than the corresponding $\mathbf{m}^{\bigwedge}_{i,t}$,$\mathbf{m}^{\bigvee}_{i,t}$. Otherwise, it is assumed that transition $s_{l,t-1}, s_{l,t}$ has never been observed in C_i, and is anomalous.

5.3 Likelihood of the Same-Category Segment

Suppose there is a sequence $\mathbf{s_i}=(s_{i,1}, s_{i,2}, \ldots, s_{i,l}, \ldots, s_{i,n_i}), 1\leq i\leq n^T$ assigned to a cluster $C_j, 1\leq j\leq k$. For an arbitrary $t \neq 0, r_{\bigwedge}\leq l - t<l<l + t\leq n_i$, a segment $\mathbf{s}^{\bigvee}_{\mathbf{i,t}} = \{s_{i,l-t}, s_{i,l-t+1}, \ldots, s_{i,l+t}\}$ is introduced, denoting the longest substring observed inside $\mathbf{s_i}$ after t-th request, and composed of the same elements $s_{i,l-t}$, such that $s_{i,l-t} = s_{i,l-t+1} = \ldots = s_{i,l+t}$. As a result, one could be interested in the likelihood of observing segment $\mathbf{s}^{\bigvee}_{\mathbf{i,t}}$ among other connections from C_j. Therefore, a joint distribution $P_j(s_{\bigvee}, n^r)$ has been introduced, with a sample space $\Omega_{s_{\bigvee},n^r}=\{(s_{\bigvee}, n^r) \in A\}, A=\{(r_{\bigwedge}, r_{\bigwedge}), \ldots, (r_{\bigvee}, r_{\bigvee})\}$. $P_j(s_{\bigvee}, n^r)$ describes a distribution of an arbitrary segment $s_{\bigvee}$ composed of requests coming from one category given a number of requests n^r.

Therefore, for each sequence $\mathbf{s_i}$ belonging to C_j, conditional probability $P_j(s_i^{\bigvee}|n_i)$ of the longest segments of the same categorical requests, given the number of requests n_i is calculated in the following way:

$$P_i(s^{\bigvee}_{i,\cdot}|n_i)= \prod_{l=r_{\bigwedge}}^{n_j} P_i(s^{\bigvee}_{i,l}|l) \tag{7}$$

Note well that while t varies from $r_{\bigwedge}$ to n_i, there will be a number of different same-category segments $\mathbf{s}^{\bigvee}_{\mathbf{i,t}}$ inside $\mathbf{s_i}$.

Subsequently, for each cluster C_j, set $\mathbf{L_j}=\{\ell_1(s_{\bigvee}, n^1), \ell_2(s_{\bigvee}, n^2), \ldots\ \ell_t(s_{\bigvee}, n^t), \ldots, \ell_{|C_i|}(s_{\bigvee}, n^{|C_i|})\}$ is introduced, where $\ell_t(s_{\bigvee}, n^t)$ denotes loglikelihoods of the longest same-element segments, observed in sequence $\mathbf{s_t}$.

In this report it is assumed that if loglikelihood $\ell_j(s_{\bigvee}, n^j)$ is smaller than $\frac{1}{2}\min \mathbf{L_i}$, then $\mathbf{s_j}$ will be marked illegitimate ($\frac{1}{2}\min \mathbf{L_j}$ is changed to $\min \mathbf{L_j}$ for softer detection approaches, see Sect. 5.4).

5.4 Soft and Hard Detection Ranges

As it is shown in the following section, regardless of a strategy the attacking hosts are programmed with, attackers do not know, and cannot predict *recent patterns of behaviour*; nor can they guess the number of connections that expressed behaviour encoded in different clusters from $\mathbf{C}$. In addition, connection distribution across clusters is similar for D^V and D^T (see Sect. 4). As a result, the attackers fail to repeat transitions encoded in D^T, and most of their zombies are assigned to higher entropy clusters. Therefore, classification of sequences from $D^V \bigcup A$ should vary for clusters with low and high entropy, and be divided

into two ranges: "soft" and "hard". As a result, an entropy range $\mathbf{H^H}=\{h_1,\ldots,h_t\}, 1\leq t<k$ is introduced for which "hard" detection technique range is applied (released to "soft" for fewer populated clusters). For the same reason, there is a corresponding range $\mathbf{H^S}=\{h_{t+1},\ldots,h_k\}$ for which "soft" detection method applies. Clearly $\mathbf{H}=\mathbf{H^H}\bigcup\mathbf{H^S}$, and $|\mathbf{H}|=|\mathbf{C}|=k$.

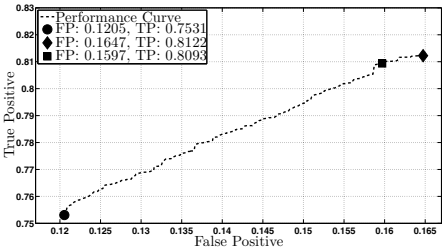
In addition, while analysing connections from $D^{V\bigcup A}$, special attention should be paid to clusters for which an increased number of connections is observed comparing to D^T. Therefore, initially the percentages $f_i^T, f_i^{V\bigcup A}, 1\leq i\leq k$ of sequences from batches D^T and $D^{V\bigcup A}$ assigned to cluster C_i are calculated. If $f_i^T<f_i^{V\bigcup A}$, then it is assumed that a big fraction of sequences from $D^{V\bigcup A}$ assigned to C_i are attacking. Otherwise, if there is a cluster $C_j, 1\leq j\leq k$ for which $f_j^T \geq f_j^{V\bigcup A}$ then it is assumed that most of the sequences from $D^{V\bigcup A}$ are legitimate.

Therefore, for $i, 1\leq i\leq k$ if entropy of a cluster C_i has been selected for $\mathbf{H^H}$ and $f_i^T<f_i^{V\bigcup A}$, then "hard" detection techniques will be used against connections from $D^{V\bigcup A}$ assigned to C_i. In addition, suppose C_j was assigned to $\mathbf{H^H}$. However, because $f_j^T \geq f_j^{V\bigcup A}$, "soft" detection measures would be used instead. It stems from the fact that distribution of connections inside D^V and D^T are similar. As a result, even though $j\in\mathbf{H^H}$ it is still reasonable to assume that most of the connections from $D^{V\bigcup A}$ inside C_j are legitimate.

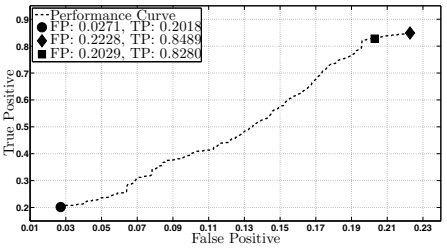
6 Application of Statistical Measurements

Given the entropy range $\mathbf{H}$ and its division into "hard" and "soft" ranges, the performance of the detection techniques is introduced.

For the first batch (i.e. the dataset with frequently-changing and validating connections), mutual information will be used to detect anomalous sequences for "soft" clusters. On the contrary, for "hard" clusters, mutual information will be supported with Mahalanobis distance. Similarly, for the other batch (i.e. the dataset with rarely-changing and validating connections), for "soft" clusters $\min\mathbf{L_j}$ is used for comparison, which is supported with mutual information and changes to $\frac{1}{2}\min\mathbf{L_j}$ for "hard" clusters. The performance of the detection method is presented in Fig. 4. Initially, h_t has been set to the smallest entropy value, and entire the cluster set $\mathbf{C}$ has been checked with "soft" detection technique. For the dataset with the frequently-changing hosts, 88% of sequences from D^V, and 75% of sequences from D^A have been correctly classified (a circle in Fig. 4a); this corresponds to 98% of sequences from D^V, and only 20% of sequences from D^A for the other dataset (a circle in Fig. 4b). Subsequently, h_t has been iteratively moved to h_k, when the whole set of clusters has been checked against "hard" technique. In this setting, 84% of sequences in D^V, and 81% of sequences in D^A from the dataset with the frequently-changing hosts have been recognised (a diamond in Fig. 4a); this corresponds to recognition of 78% of sequences in D^V, and 85% of sequences in D^A for the other dataset (a diamond in Fig. 4b). Note well that Fig. 4 shows only the performance of the algorithm for possible $h_t\in\mathbf{H}$.



(a) Performance curve for $D^{V \bigcup A}$ composed of validating and frequently-changing hosts.



(b) Performance curve for $D^{V \bigcup A}$ composed of validating and rarely-changing hosts.

Fig. 4. Performance of the detection measurements against two attacking strategies: frequently-changing (4a), and rarely-changing(4b)

The best detection setting against frequently-changing zombies recognises 81% of attacking and 84% legitimate sequences , for h_t=h_{292}=3.8630 (a square in Fig. 4a). The best performance against the rarely-changing hosts has been observed for h_t=h_{104}=2.8563, that is when 80% of legitimate and 83% of attacking zombies have been recognised (a square in Fig. 4b). Threshold h_t is larger for the frequently-changing zombies, because of the increased number of these zombies being assigned to higher-entropy clusters. The opposite is observed for the rarely-changing zombies, as their sequences have been generated to be similar to sequences from D^V. The time needed for processing of $n^{V \bigcup A} = 2n^V = 2 \cdot 82,518 = 165,036$ sequences takes roughly 10 minutes, and is similar to [17].

7 Related Work

Xie developed IDS based on Hidden-Markov Models [26],[27]. Their model, rather than considering categories, uses objects stored at a website (i.e. images, scripts etc.), and frequency of requested objects. The rationale behind using Hidden-Markov Models for web stems from its successful application for wireless devices [1], and turns us to be successful for caching, pre-fetching and next page prediction [12]. Therefore, for web application it is reasonable to consider sequences of requests as product of conditional probabilities as in Sect. 2.6.

Stevanovic has introduced clustering of web sessions based on Self-Organising Maps [21], [22]. In addition, Adaptive Reasoning Theory neural network has been chosen to provide difference in variances among: human visitors, well-behaved web crawlers, malicious crawlers and unknown visitors. The authors have managed to recognise 95% human-generated sequences. In addition, K-means clustering used successfully for sequence-independent analysis [13].

In [24], the authors have developed a system based on Mahalanobis distance for payload analysis. Moreover, known variations of HTTP-GET attack have been tested against the trained system with linear discriminant analysis that assigns sequences to the legitimate set or any of the known attacks.

Mao et. al. [17] have developed a sequence-based detection method for different attacks. Their methods is based on graph-theoretical representation of system events, and takes on a sample of known instances of attacks as well. The accuracy performance of the system ranges between 80% to 90%. Also, the authors have showed that their system can process 300,000 sequences in 20 minutes. Moreover, there are many sound scientific publications developing detection methods against known attacks instances [2],[6],[9],[25].

8 Result Discussion and Future Work

In this paper a detection method has been presented against two types of attacking hosts: frequently-changing and rarely-changing. It has been shown that the introduced clustering algorithm allows for detection of either of the attacking attempts.

We have shown a novel detection technique against HTTP-GET attacks, based on various statistical distances and using entropy-minimisation for clustering. The main advantage of our work should be attributed to considerably good results, given that the frequency of arriving request is not considered, and only categorical interest is taken into consideration. In addition, no a priori information is assumed for attacking behaviour. Moreover, the method works similarly (i.e. attains similar detection rate) for two scenarios of attacks. Also, categories could be files or even hyper-links, that is why the clustering method is not limited to web categories only.

The *only* techniques for improving performance of classification are either to extend feature space or improve dataset decomposition, or both [14]. We still refrain from temporal analysis (time-independent assumptions make passing detection test harder for attackers), and believe that better data clustering approach can be obtained with cross-entropy approach [4]. Using cross-entropy will allow to replace iterative clustering method defined in Sect. 2, with adaptive optimisation settings. In addition, because there are many sequences inside D^T and D^V that follow almost uniform distribution of requests (indicating that human agents can follow uniform pattern of requests, as it has been indicated in [21]), it is really difficult to find a right measurement for clusters spread across different entropy range. Therefore, finding a better, attack-independent measurements is scheduled for our future work as well.

Acknowledgement. The authors would like to thank the anonymous referees for their helpful comments and suggestions to improve this work.

References

1. Anderson, C.R., Domingos, P., Weld, D.S.: Adaptive web navigation for wireless devices. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence, vol. 2, pp. 879–884. Morgan Kaufmann Publishers Inc., San Francisco (2001)

2. Ariu, D., Tronci, R., Giacinto, G.: Hmmpayl: An intrusion detection system based on hidden markov models. Computers and Security 30(4), 221–241 (2011)
3. Barbará, D., Li, Y., Couto, J.: Coolcat: an entropy-based algorithm for categorical clustering. In: Proceedings of the Eleventh International Conference on Information and knowledge Management, CIKM 2002, pp. 582–589. ACM, New York (2002)
4. de Boer, P.-T., Kroese, D., Mannor, S., Rubinstein, R.: A tutorial on the cross-entropy method. Annals of Operations Research 134, 19–67 (2005), doi:10.1007/s10479-005-5724-z
5. Denning, D.E.: An intrusion-detection model. IEEE Trans. Softw. Eng. 13(2), 222–232 (1987)
6. Ingham, K.L., Somayaji, A., Burge, J., Forrest, S.: Learning dfa representations of http for protecting web applications. Computer Networks 51(5), 1239–1255 (2007); From Intrusion Detection to Self-Protection
7. Jalali, M., Mustapha, N., Nasir Sulaiman, M., Mamat, A.: Webpum: A web-based recommendation system to predict user future movements. Expert Systems with Applications 37(9), 6201–6212 (2010)
8. Jung, J., Krishnamurthy, B., Rabinovich, M.: Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites. In: Proceedings of the 11th International Conference on World Wide Web, WWW 2002, pp. 293–304. ACM, New York (2002)
9. Kruegel, C., Vigna, G., Robertson, W.: A multi-model approach to the detection of web-based attacks. Comput. Netw. 48(5), 717–738 (2005)
10. Kullback, S., Leibler, R.A.: On Information and Sufficiency. The Annals of Mathematical Statistics 22(1), 79–86 (1951)
11. Kumar, P., Radha Krishna, P., Bapi, R.S., Kumar De, S.: Rough clustering of sequential data. Data and Knowledge Engineering 63(2), 183–199 (2007)
12. Lee, C.-H., Lo, Y.L., Fu, Y.-H.: A novel prediction model based on hierarchical characteristic of web site. Expert Systems with Applications 38(4), 3422–3430 (2011)
13. Lee, S., Kim, G., Kim, S.: Sequence-order-independent network profiling for detecting application layer ddos attacks. EURASIP Journal on Wireless Communications and Networking 2011(1), 50 (2011)
14. Lee, W., Stolfo, S.J.: A framework for constructing features and models for intrusion detection systems. ACM Trans. Inf. Syst. Secur. 3, 227–261 (2000)
15. Lee, W., Xiang, D.: Information-theoretic measures for anomaly detection. In: IEEE Symposium on Security and Privacy, pp. 130–143 (2001)
16. Li, T., Ma, S., Ogihara, M.: Entropy-based criterion in categorical clustering. In: Proceedings of the Twenty-First International Conference on Machine Learning, ICML 2004, pp. 68–75. ACM, New York (2004)
17. Mao, C.-H., Pao, H.-K., Faloutsos, C., Lee, H.-M.: Sbad: Sequence based attack detection via sequence comparison. In: PSDML, pp. 78–91 (2010)
18. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal 27 (1948)
19. Speiser, M., Antonini, G., Labbi, A., Sutanto, J.: On nested palindromes in clickstream data. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2012, pp. 1460–1468. ACM, New York (2012)
20. Srivatsa, M., Iyengar, A., Yin, J., Liu, L.: Mitigating application-level denial of service attacks on Web servers: A client-transparent approach. ACM Trans. Web 2, 15:1–15:49 (2008)

21. Stevanovic, D., Vlajic, N., An, A.: Unsupervised Clustering of Web Sessions to Detect Malicious and Non-malicious Website Users. Procedia CS 5, 123–131 (2011)
22. Stevanovic, D., Vlajic, N., An, A.: Detection of malicious and non-malicious website visitors using unsupervised neural network learning. Applied Soft Computing (2012)
23. Strehl, A., Ghosh, J., Mooney, R.: Impact of Similarity Measures on Web-page Clustering. In: Proceedings of the 17th National Conference on Artificial Intelligence: Workshop of Artificial Intelligence for Web Search (AAAI 2000), Austin, Texas, USA, July 30-31, pp. 58–64. AAAI (July 2000)
24. Tan, Z., Jamdagni, A., He, X., Nanda, P., Liu, R.P., Jia, W., Yeh, W.-C.: A Two-Tier System for Web Attack Detection Using Linear Discriminant Method. In: Soriano, M., Qing, S., López, J. (eds.) ICICS 2010. LNCS, vol. 6476, pp. 459–471. Springer, Heidelberg (2010)
25. Ulmer, C., Gokhale, M., Gallagher, B., Top, P., Eliassi-Rad, T.: Massively parallel acceleration of a document-similarity classifier to detect web attacks. Journal of Parallel and Distributed Computing 71(2), 225–235 (2011); Data Intensive Computing
26. Xie, Y., Yu, S.-Z.: A Novel Model for Detecting Application Layer DDoS Attacks. In: Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2006), vol. 2, pp. 56–63. IEEE Computer Society, Washington, DC (2006)
27. Xie, Y., Yu, S.-Z.: Monitoring the application-layer DDoS attacks for popular websites. IEEE/ACM Trans. Netw. 17, 15–25 (2009)

A Generic Algebraic Model for the Analysis of Cryptographic-Key Assignment Schemes

Khair Eddin Sabri[1] and Ridha Khedri[2]

[1] Department of Computer Science
King Abdulla II School for Information Technology
The University of Jordan
k.sabri@ju.edu.jo
[2] Department of Computing and Software
Faculty of Engineering
McMaster University
khedri@mcmaster.ca

Abstract. One of the means to implement information flow policies is by using a cryptographic approach commonly referred to as key assignment schemes. In this approach, information is made publicly available to users but in an encrypted form. Then, keys are assigned to users such that each key reveals a specified part of the information. Usually the distribution of keys follows a predefined scheme that specifies the ability of users to reveal information.

In this paper, we present an algebraic approach based on idempotent commutative semirings to define, specify, and analyse key assignment schemes. Then, we illustrate its usage on two key assignment schemes selected from the literature. Also, we propose amendments to the studied schemes to extend their scopes. The proposed generic algebraic approach enables the verification of security properties at an abstract level in systems that use key assignment schemes. The verification takes into consideration the algebraic properties of schemes, and the considered relationships among the assigned keys. Then, it enables the verification of the secrecy properties of the system through algebraic calculations. All the calculations can be automated using a theorem prover such as Prover9.

1 Introduction

The confidentiality of information is an important aspect considered in many organizations. Policies specifying users access to information ensure the confidentiality of information. Then, they are embodied in access control mechanisms enabling organizations to restrict information access to legitimate users only.

Another way to achieve the confidentiality of information is through the use of cryptography where objects (e.g., files) are encrypted. Encryption and decryption keys are used in a such a way that each key decrypts parts of the objects and reveals the part of the information that is intended for the key owner. For that reason, keys are distributed to users to allow each one to decrypt the objects

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 62–77, 2013.

and reveal the information she is allowed to reveal. In [14,15], we classify key distribution schemes into key-based schemes or object-based schemes. A *key-based scheme* focuses on the relationship between keys and their use to decrypt objects, while an *object-based scheme* focuses on objects and the required conditions to decrypt each one of them. In this paper, we consider only key-based schemes.

In a *key-based scheme*, objects are organized in a hierarchical structure. They are classified into security levels denoted by c such that, for some i and j natural numbers, $c_i \leq c_j$ indicates that the security level c_j is more sensitive than the security level c_i. Therefore, a user having an access to an information classified as c_j can also have an access to an information classified as c_i. Usually the relation $\leq$ is a partial order relation (i.e., $\leq$ is transitive, reflexive, and antisymmetric). This information accessibility that is based on the security levels is commonly referred to as the *simple security property* or *hierarchical access control*.

Users should hold the appropriate keys to get the authorized information at their levels and the levels below them in the hierarchy. There can be different approaches to assign keys to users. One approach requires assigning each user several keys such that each key reveals the information of one security class. This approach is not efficient and adds overhead to the user. Another approach requires having several copies of the information encrypted with different keys. This approach enhances the chances of a crypto-analysis to reveal the confidential information. A better approach is to let each user handle one key and to have only one copy of an encrypted information.

1.1 Motivation and Contribution

Several techniques exist in the literature to handle key assignment [1,2,9,16]. However, several of them have been found to be flawed or very weak in preserving the secrecy of confidential information as given in [6]. Crampton et al. [6] advocate the adoption of a generic model for key assignment schemes that would be used for evaluating proposals for key assignment schemes. We think that formal analysis of proposed key assignment schemes is another means for asserting their dependability. A mathematical framework for the verification of a proposed scheme enables us to mathematically assert, within the boundary of the hypothesis of our mathematical model, its correctness in preserving the confidentiality of the information. Having an abstract view of these schemes gives a better understanding of key assignment. Also, it allows identifying the relationships among them and helps in proposing more general schemes.

In this paper, we propose a generic model for the specification and analysis of cryptographic-key assignment schemes. The model is articulated in Definition 2. Based on this model, we analyse two representative schemes: the Akl-Taylor key assignment [1] scheme, and a scheme based on the Chinese remainder theorem [5]. We also show how they can be generalised and extended to assign more than one key to a security class. We also illustrate the automation of the analysis of systems that use key assignment schemes using the Prover9 theorem prover.

1.2 Organisation

In Section 2, we give the required mathematical background. In Section 3, we define a key assignment scheme. Then, we present its usage in the specification and analysis of key assignment schemes. In Section 4, we report on the schemes for cryptographic-key assignment found in the literature. Finally, we present a brief discussion and conclude in Section 5.

2 Mathematical Background

The algebraic key-structure presented in this section is used in [14,15] to capture the properties of cryptographic keys. The idea is while cryptographers examine keys from the perspective of a cryptographic system, the algebraic key-structure captures the algebraic properties that cryptographic keys in general satisfy. Within a specific cryptographic system, the keys might have additional properties than the common properties satisfied by keys in general. We will elaborate on this issue in Section 3. A key is thought of as a piece of information that determines the output of a cryptographic algorithm or cipher. In [14,15], other structures are proposed to model ciphers, combinations of keys and ciphers, and messages. The operators of the algebraic structure are used to represent functions on keys, ciphers, and messages. For example, a binary operator of the key-structure is used to represent combining keys. An inverse operator of the cipher structure is used to specify encryption or decryption depending on the context. In this section, we present the mathematical background needed to specify keys.

Let $+$ and $\cdot$ be two binary operators. The operator $+$ is associative iff $\forall(x, y, z \mid x, y, z \in S : x + (y + z) = (x + y) + z)$. It is commutative iff $\forall(x, y \mid x, y \in S : x + y = y + x)$. The constant 0 is said to be the left and right identity of $+$ iff $\forall(x \mid x \in S : 0 + x = x + 0 = x)$. The operator $+$ is idempotent iff $\forall(x \mid x \in S : x + x = x)$. We say that $\cdot$ distributes over $+$ on both the left and right iff $\forall(x, y, z \mid x, y, z \in S : x \cdot (y + z) = x \cdot y + x \cdot z)$ and $\forall(x, y, z \mid x, y, z \in S : (x + y) \cdot z = x \cdot z + y \cdot z)$.

A semigroup is an algebraic structure $\mathcal{A} = (S, \cdot)$, where S is a set and $\cdot$ is an associative binary operator. If the operator $\cdot$ is also commutative, we call $\mathcal{A}$ a commutative (Abelian) semigroup. Furthermore, if the operator $\cdot$ is idempotent, we call $\mathcal{A}$ an idempotent commutative semigroup. Let $S \neq \emptyset$ be a set and $+$ and $\cdot$ binary operations on S. Then $(S, +, \cdot)$ is called a semiring if $(S, +)$ is a commutative semigroup, $(S, \cdot)$ is a semigroup, and $\cdot$ distributes over $+$ on both the left and right. If the semigroup $(S, \cdot)$ has a neutral element 1_s, we call 1_s the identity of the semiring $(S, +, \cdot)$. If the semigroup $(S, +)$ has a neutral element 0_s, we call it the zero of the semiring $(S, +, \cdot)$. We call 0_s the multiplicatively absorbing if 0_s is absorbing in $(S, \cdot)$ i.e., $\forall(x \mid x \in S : 0_s \cdot x = x \cdot 0_s = 0_s)$. If $(S, +)$ is an idempotent semigroup, we call $(S, +, \cdot)$ an additively idempotent semiring. If $(S, \cdot)$ is a commutative semigroup, we call $(S, +, \cdot)$ a commutative semiring. If $(S, +, \cdot)$ is an additively idempotent semiring, then there exists an ordering

relation such that for $a, b \in S$ we have $a \leqslant b \iff a + b = b$. The relation $\leqslant$ is referred to as the natural ordering relation associated to the semiring.

Let C be a set. A partial order (or order) on C is a binary relation $\prec$ on C such that, for all $x, y, z \in C$, (1) $x \prec x$, (2) $x \prec y \ \wedge \ y \prec x \implies x = y$, and (3) $x \prec y \ \wedge \ y \prec z \implies x \prec z$.

A set equipped with a partial order is called an ordered set, partially ordered set, or poset. C is said to be a chain if, for all $x, y \in C$, either $x \prec y$ or $y \prec x$. When $(C, \prec)$ satisfies only Properties (1) and (3), but not (2), we say that it is a pre-ordered set (or quasi-ordered set). For a pre-ordered set $(P, \prec)$, its dual $(P, <)$ is defined as for all x, y, we have $x < y \overset{\text{def}}{\iff} y \prec x$.

Definition 1 (Key-Structure). *Let* $\mathcal{K} \overset{\text{def}}{=} (K, +_k, *_k, 0_k)$ *be an algebraic structure that is an additively idempotent commutative semiring with a multiplicatively absorbing zero* 0_k*. We call* $\mathcal{K}$ *a key-structure.*

The structure $(K, +_k, *_k, 0_k)$ can be seen as the structure of keys where K is a set of keys. The operators $+_k$ and $*_k$ can be seen as combining keys. The difference between them is that the $+_k$ operator combines keys in such a way that only *one* key is used to encrypt/decrypt one unit of plaintext/ciphertext. While the $*_k$ operator combines keys which are its arguments in such a way to use them simultaneously to encrypt or decrypt one unit. The key 0_k can be perceived as a special key that is not suitable neither for decryption nor for encryption. If the combination of two keys is not appropriate, one writes $k_1 *_k k_2 = 0_k$. Caesar cipher uses the key c_i that shifts the alphabet of the text i mod 26 positions. Vigenère cipher uses a table that contains all the 26 possible shifting of the alphabets to encrypt/decrypt a letter in the text, we choose one row only from the table. This selection can be represented through $+_k$. Therefore, we can represent the table as $k_v = c_0 +_k c_1 +_k \cdots +_k c_{25}$. Vigenère cipher uses also a keyword k_w to select the appropriate row of the table. Both the table and the keyword should be used together for the encryption or decryption of an alphabet in the text. We can represent that as $k_w *_k k_v$.

The operators of the key-structure can also be used to specify a generalisation of the RSA cipher that is introduced by Boyd [3]. This technique is based on distributing a set of keys to users such that all the keys should be combined together to decrypt a cipher. The combination of two keys (k_1, m) and (k_2, m) is given by Boyd [3] as $(k_1 \times k_2, m)$. This combination is associative and commutative and can be represented using the operator $*_k$ as $(k_1, m) *_k (k_2, m) \triangleq (k_1 \times k_2, m)$.

A unary operator, called an inverse, is usually defined in the literature on keys to relate a key used in encryption to that used in decryption. However, our representation of keys is slightly different from the ones found in the literature. We include all the information required for both encryption and decryption in one key. For example, the RSA key involves a public key for encryption (e, n) and a private key (d, n) for decryption. In the key, n is the multiplication of two large prime numbers (e.g., p and q), and e and d satisfy the congruence relation $ed \equiv 1 \pmod{\varphi(n)}$ where $\varphi(n) = (p-1)(q-1)$. We write an RSA key as one key (e, d, n), where only one part of the key is required for encryption

(i.e., (e, n)) or decryption (i.e., (d, n)). This representation avoids the use of an inverse operator related to the cipher as our aim is to focus on the key-structure without taking encryption and decryption into consideration. There are several models of the key-structure presented in [14,15].

3 Key Assignment Schemes

In this section, we present an algebraic model for key assignment schemes and show its use in analysing key assignment schemes.

Definition 2 (Key assignment scheme). *We call a key-assignment scheme the system $(\mathcal{K}, C, \prec, a)$, where:*

- *$\mathcal{K}$ is a key-structure,*
- *$(C, \prec)$ is a poset, and*
- *$a \subseteq K \to C$ is a surjective (onto) function.*

C and a are respectively identified as the set of security classes, and the assignment function. The poset $(C, \prec)$ is said to be the poset of the scheme $\mathcal{S}$.

The function a needs to be surjective to ensure that each element in C (i.e., each security class) is assigned a key. Usually, keys are assigned to users, and users at their turn are assigned to security classes. Then, for all $x, y \in U$, $x \prec_u y$ if and only if the security class of x is lower than the security class of y. The structure $(U, \prec_u)$ is a poset. Therefore, we need only to consider the security classes as there is an order isomorphism between $(C, \prec)$ and $(U, \prec_u)$, which is the map s from U onto C such that $x \prec_u y$ if and only if $s(x) \prec s(y)$. The map $s(x)$ gives the security class of the user x. We assume that each class contains at least one user. Therefore, the two above posets are essentially the same as they are isomorphic. We need to have a as function to ensure that no key gets assigned to more than one security class, which would make the assignment scheme nonsecure. Obviously, a class can be assigned several keys.

On the domain of the key-assignment function a, we define a relation that we denote by $\prec_d$. We read $k_1 \prec_d k_2$ as part of the information that can be revealed by using k_1 can be also revealed by using k_2. Obviously, $(\mathsf{dom}\,(a), \prec_d)$ is a pre-order (quasi-order) as it is reflexive, transitive, and not necessarily antisymmetric. Next, we present this relation within the key structure.

The structure $\mathcal{K}$ is an additively idempotent commutative semiring. Therefore, it has a natural order relation $\leq$ inherent to it (i.e., $\forall(x, y \mid x, y \in K : x \leq y \iff x +_k y = y)$). One can read $k_1 \leq k_2$ as the key k_1 is a sub-key of the key k_2. The meaning of "sub-key" would depend on the interpretation of the operation $+_k$ on keys. For example, in the Vigenère table, each row in that table can be considered as a subkey of the whole table. We find in [8] a preorder relation based on $\leq$. We denote it by $\sqsubseteq$ and it is defined as:

$$a \sqsubseteq b \overset{\text{def}}{\iff} \exists(c \mid c \in K : a \leq b *_k c).$$

The relation $\sqsubseteq$ is a pre-order (the proof can be found in [8]). Therefore, it can be taken as the relation $\prec_d$. In the sequel, we will discuss and illustrate its usage and that of its dual as the relation $\prec_d$.

The following proposition holds for any additively idempotent commutative semiring with a multiplicatively absorbing zero 0 and an identity 1. Therefore, it holds for any key structure with an identity 1_k. There are many key structures that have an identity key (i.e., has no effect when combining it with another key). For example, a key in RSA cipher can be represented as (k_1, m) and combining the keys (k_1, m) and (k_2, m) can be as $(k_1 \times k_2, m)$. Therefore, we can have the identity key as $(1, m)$. We show the use of this proposition in the analysis of key assignment schemes in the next subsections.

Proposition 1 (e.g., [8]). *Let $\mathcal{K} = (K, +_k, *_k, 0_k, 1_k)$ be a key structure with an identity 1_k. Let $k_1, k_2 \in K$ be keys. We have:*

1. $k_1 \leq_k k_2 \implies k_1 \sqsubseteq k_2$
2. $k_1 *_k k_2 \sqsubseteq k_2$
3. $k_1 \sqsubseteq k_2 \implies k_1 +_k k_3 \sqsubseteq k_2 +_k k_3$
4. $k_1 \sqsubseteq k_2 \implies k_1 *_k k_3 \sqsubseteq k_2 *_k k_3$
5. $k \sqsubseteq 1_k$

Identity (1) indicates that the ordering relation between keys (i.e., sub-key relation) implies the relation $\sqsubseteq$ between keys. Identity (2) relates the combining keys operator to the relation $\sqsubseteq$. Identities (3) and (4) are standard isotony laws with respect to combining keys. Identity (5) relates the 1_k keys to the relation $\sqsubseteq$.

Definition 3. *Let $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{K}, C, \prec, a)$ be a key-assignment scheme. Given a key-derivation relation $\prec_d$ defined on $\mathsf{dom}\,(a)$, the scheme $\mathcal{S}$ is said to be cluster-secure with regard to $\prec_d$ iff $\forall (k_i, k_j \mid k_i, k_j \in \mathsf{dom}\,(a) \wedge (k_i \neq k_j) \wedge (a(k_i) \prec a(k_j)) : \neg(k_j \prec_d k_i))$.*

Definition 3 states that a key-assignment scheme is a cluster secure if, within a closer (i.e., connected subgraph) of the poset of the scheme, low class keys cannot reveal the information of its higher classes.

Definition 4. *Let $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{K}, C, \prec, a)$ be a key-assignment scheme. Given a key-derivation relation $\prec_d$ defined on $\mathsf{dom}\,(a)$, the scheme $\mathcal{S}$ is said to be class-secure with regard to $\prec_d$ iff C is a chain and $\mathcal{S}$ is cluster-secure.*

Definition 4 states that if a key-assignment scheme which all its security classes are comparable with the relation $\prec$ and that is a cluster secure, then it is class-secure. The fact that $(C, \prec)$ is a chain leads to having a poset with one connected graph and all the classes can be compared.

Definition 5. *Let $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{K}, C, \prec, a)$ be a key-assignment scheme. Given a key-derivation relation $\prec_d$ defined on $\mathsf{dom}\,(a)$, the scheme $\mathcal{S}$ is said to be user-secure with regard to $\prec_d$ iff $\forall (k_i, k_j \mid k_i, k_j \in \mathsf{dom}\,(a) \wedge (k_i \neq k_j) : \neg(k_j \prec_d k_i))$.*

Definition 5 states that a key-assignment scheme that contains independent keys such that no key can reveal the information that can be revealed from other keys is called user-secure.

Proposition 2. *Let* $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{K}, C, \prec, a)$ *be a key-assignment scheme. If* $\mathcal{S}$ *is user-secure, then it is cluster-secure.*

Proof. $\mathcal{S}$ is user-secure

$\iff$ $\langle$ Definition 5 $\rangle$

$\forall(k_i, k_j \mid k_i, k_j \in \mathsf{dom}\,(a) \wedge (k_i \neq k_j) : \neg(k_i \prec_d k_j))$

$\implies$ $\langle$ $((a(k_i) \prec a(k_j)) \vee \neg(a(k_i) \prec a(k_j)) \iff \mathsf{true})$. Range split. Weakening (i.e., $(p \wedge q) \implies p$ for p, q predicates) $\rangle$

$\forall(k_i, k_j \mid k_i, k_j \in \mathsf{dom}\,(a) \wedge (k_i \neq k_j) \wedge (a(k_i) \prec a(k_j)) : \neg(k_i \prec_d k_j))$

$\iff$ $\langle$ Definition 3 $\rangle$

$\mathcal{S}$ is cluster-secure

We have the other direction of the implication only when every class contains only one user.

Lemma 1. *Let* $\mathcal{S} \stackrel{\text{def}}{=} (\mathcal{K}, C, \prec, a)$ *be a cluster-secure key-assignment scheme. We have*

$$\forall(k_i, k_j \mid k_i, k_j \in \mathsf{dom}\,(a) \wedge (k_i \neq k_j) : a(k_i) \prec a(k_j)) \implies \forall(k_i, k_j \mid k_i, k_j \in \mathsf{dom}\,(a) \wedge (k_i \neq k_j) : \neg(k_i \prec_d k_j))$$

Proof. It is obvious due to monotonic $\forall$-body rule.

3.1 Specifying the Akl-Taylor Technique

The Akl-Taylor [1] technique is a pioneer technique in key assignment by deriving one key from another. The technique assigns to each user a key k_i such that $k_i = \kappa^{t_i} \pmod m$ where κ is a private number, m is a public number that is the product of two large prime numbers, and t_i is a public number formed from a multiplication of prime numbers. A key k_j can be derived from k_i iff t_j is divisible by t_i as $k_i^{t_j/t_i} = (\kappa^{t_i})^{t_j/t_i} \pmod m = \kappa^{t_j} \pmod m = k_j$. For example, let $m = 11 \times 17 = 187$ (we use small prime numbers instead of large ones for illustration only) and $\kappa = 13$ such that the greatest common divisor between m and κ is 1. We can assign to a first user the public number $3 \times 5 \times 7 = 105$ and therefore, the key becomes $13^{105} \pmod{187} = 98$. We can assign to a second user a public number that divides 105 such as $5 \times 7 = 35$ and therefore, the key becomes $13^{35} \pmod{187} = 21$. We can assign to a third user a public number that divides 35 such as 7 and therefore, the key becomes $13^7 \pmod{187} = 106$. The key 106 can be used to derive the keys 98 and 21 as $106^{15} \pmod{187} = 98$ and $106^5 \pmod{187} = 21$. Also, the key 21 can be used to derive the key 98 as $21^3 \pmod{187} = 98$.

The main idea behind the Akl-Taylor technique [1] is that each key is represented as $k_i = \kappa^{t_i} \pmod m$ where κ is a private data, m is public data formed by multiplying two large prime numbers, and t_i is a product of a set of (distinct) prime numbers. Therefore, for a given m, we have $\frac{\log k_i}{\log \kappa} = t_i$. This equation

indicates that when κ is given, a key is determined by the exponent t_i which is the product of a set of distinct prime numbers. The server that distributes the keys, determines the values of κ and keeps it private. In analysing the Akl-Taylor scheme, Crampton et al. [6] as well find that it is sufficient to represent a key as a set of primes.

Generalisation of the Akl-Taylor Technique
Once κ is fixed, the exponent t_i determines the key. One can generalize the Akl-Taylor technique by perceiving keys as sets of products of distinct elements of a given set of prime numbers that we denote by $\mathbb{N}_p$. For instance, a key having an exponent $t_i = 2 \times 3 \times 7$ can be represented as $\{\{2, 3, 7\}\}$ for a given κ and m. This representation is valid due to the commutativity of the multiplication (i.e., the order does not matter as in a set there is no order among its elements). The use of a set of sets allows us to represent a set of keys such that each internal set corresponds to a key of the Akl-Taylor technique. Let $P \stackrel{\text{def}}{=} \{p_1 \times \cdots \times p_n \mid \exists(p_1 \cdots p_n \mid p_i \in \mathbb{N}_p : \forall(p_i, p_j \mid p_i, p_j \in \mathbb{N}_p : i \neq j \implies p_i \neq p_j))\}$. We define a function *rep* that takes an exponent of a key and returns its set representation.

$$\begin{aligned} &\text{rep} : P \to \mathcal{P}(\mathcal{P}(\mathbb{N}_p)) \\ &\text{rep}(p_1 \times p_2 \times \cdots \times p_n) \stackrel{\text{def}}{=} \{\{p_1, p_2, \cdots, p_n\}\} \ . \end{aligned}$$

We note that this function is a bijection. Also, if we assign to an agent a set of keys which their exponents correspond to $t_i = 2 \times 3 \times 7$ and $t_j = 11 \times 17 \times 2$, we can say that the agent has the set of keys represented by $\{\{2, 3, 7\}, \{2, 11, 17\}\}$. Let $\mathbb{FF} \stackrel{\text{def}}{=} (\mathcal{P}(\mathcal{P}(\mathbb{N}_p)), +_k, *_k, 0, 1)$ be a structure where the elements of S are sets of subsets of $\mathbb{N}_p$ and where $*_k$ and $+_k$ are defined as follows:

$$\begin{aligned} &*_k : \mathcal{P}(\mathcal{P}(\mathbb{N}_p)) \times \mathcal{P}(\mathcal{P}(\mathbb{N}_p)) \to \mathcal{P}(\mathcal{P}(\mathbb{N}_p)) \\ &A *_k B \stackrel{\text{def}}{=} \{a \cup b : a \in A, b \in B\} \ . \end{aligned}$$

$$\begin{aligned} &+_k : \mathcal{P}(\mathcal{P}(\mathbb{N}_p)) \times \mathcal{P}(\mathcal{P}(\mathbb{N}_p)) \to \mathcal{P}(\mathcal{P}(\mathbb{N}_p)) \\ &A +_k B \stackrel{\text{def}}{=} A \cup B \ , \end{aligned}$$

The structure $\mathbb{FF} \stackrel{\text{def}}{=} (\mathcal{P}(\mathcal{P}(\mathbb{N}_p)), +_k, *_k, 0, 1)$ is a key structure with an identity 1_k. The proof of this claim invokes basic set theory properties. Since $\mathbb{FF}$ is a model for our key-structure, the system $(\mathbb{FF}, C, \prec, a)$ is a key assignment scheme. We can define on it, the relations $\leq$ and $\sqsubseteq$ introduced at the beginning of Section 3. The relation $\leq$ is identical to $\subseteq$. The definition of $\sqsubseteq$ becomes $a \sqsubseteq b \stackrel{\text{def}}{\iff} \exists(c \mid c \in \mathcal{P}(\mathcal{P}(\mathbb{N}_p)) : a \subseteq b *_k c)$. We can define $k_1 \prec_d k_2 \stackrel{\text{def}}{\iff} k_1 \sqsubseteq k_2$. In this case $\prec_d$ is $\sqsubseteq$. We will see in the next example that $\prec_d$ is the dual of $\sqsubseteq$. It simply depends on the model of the key structure that we are considering.

The system $(\mathbb{FF}, C, \prec, a)$ presents a generalization of the Akl-Taylor technique. The key in our case is not a single key but a set of keys e.g., $\{\kappa^{2\times3}, \kappa^{5\times7}\}$. In the

Akl-Taylor technique, $(C, \prec)$ has to be a tree, while in our framework it can be a forest. Therefore, for dealing with more than a tree structure and for handling more than one key per user, the Akl-Taylor technique is a special case of the one we propose. We may need this generalization if a user is involved in more than one scheme.

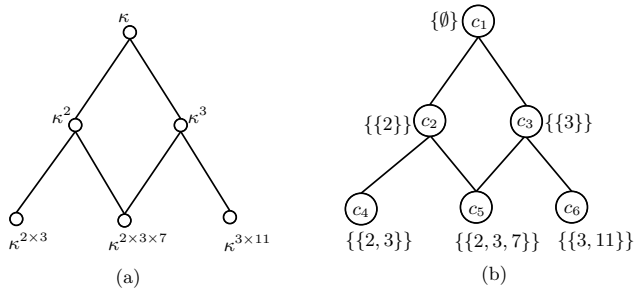


Fig. 1. An example of the Akl-Taylor scheme and its equivalent scheme

Example 1. Figure 1 shows an example of the Akl-Taylor scheme and its representation using our mathematical structure. In the system $(\mathbb{FF}, C, \prec, a)$, $\mathbb{FF}$ is defined as above, $C = \{c_1, c_2, c_3, c_4, c_5, c_6\}$ such that $c_4 \prec c_2$, $c_5 \prec c_2$, $c_5 \prec c_3$, $c_6 \prec c_3$, $c_2 \prec c_1$, $c_3 \prec c_1$, and the function a is defined as $a = \{(\emptyset, c_1), (\{\{2\}\}, c_2), (\{\{3\}\}, c_3), (\{\{2,3\}\}, c_4), (\{\{2,3,7\}\}, c_5), (\{\{3,11\}\}, c_6)\}$. For instance, the key $\kappa^{2\times3}$ is derived from κ^2. Indeed,

$$
\begin{array}{ll}
& \kappa^{2\times3} \prec_d \kappa^2 \\
\iff & \quad \langle \text{ A key is determined by its exponent \& } k_1 \text{ is derived from } k_2 \\
& \quad \text{iff } k_1 \sqsubseteq k_2, \text{ and } \frac{\log k_i}{\log \kappa} = t_i \ \rangle \\
& \mathrm{rep}(2 \times 3) \sqsubseteq \mathrm{rep}(2) \\
\iff & \quad \langle \text{ Definition of the function rep, and Definition of } \sqsubseteq \ \rangle \\
& \exists (c \mid c \in \mathcal{P}(\mathbb{N}_p) \ : \ \{\{2,3\}\} \leq \{\{2\}\} *_k c\,) \\
\iff & \quad \langle \text{ Definition of } x \leq y \text{ for } x \text{ and } y \text{ elements of an idempotent} \\
& \quad \text{commutative semiring } \rangle \\
& \exists (c \mid c \in \mathcal{P}(\mathbb{N}_p) \ : \ \{\{2,3\}\} +_k \{\{2\}\} *_k c = \{\{2\}\} *_k c\,) \\
\iff & \quad \langle \text{ Definition of } +_k \text{ on the structure } \mathbb{FF} \ \rangle \\
& \exists (c \mid c \in \mathcal{P}(\mathbb{N}_p) \ : \ \{\{2,3\}\} \cup \{\{2\}\} *_k c = \{\{2\}\} *_k c\,) \\
\impliedby & \quad \langle \ c = \{\{3\}\} \in \mathcal{P}(\mathbb{N}_p), \text{ and the definition of } *_k \text{ on the structure} \\
& \quad \mathbb{FF} \ \rangle \\
& \exists (c \mid c \in \mathcal{P}(\mathbb{N}_p) \ : \ \{\{2,3\}\} \cup \{\{2,3\}\} = \{\{2,3\}\}\,) \\
\iff & \quad \langle \text{ Idempotence of } \cup,\ c \in \mathcal{P}(\mathbb{N}_p), \text{ and } \exists (c \mid : \mathsf{true}\,) \equiv \mathsf{true} \ \rangle \\
& \mathsf{true}
\end{array}
$$

One can verify that the given example is cluster-secure by checking that $(c_i \prec c_j \implies \neg(a(c_i) \sqsubseteq a(c_j)))$.

From this example, we observe that to apply the general theory of the framework, we had only to perform the following tasks: (1) give a detailed meaning

(an interpretation) to each of the operators of the key structure, (2) define the order of the poset of the scheme, and (3) articulate the key-assignment function. The tasks (2) and (3) are specific to the application that the specifier is considering, while (1) is to set up the concrete mathematical model of the key structure used by the specifier. All the rest of the theory remains valid and can be used automatically. The next example that we give when we discuss specifying the Chen-Chung technique requires the specifier to perform the same three tasks.

Algebraic Analysis Based on Proposition 1
We use Proposition 1 to obtain interesting properties regarding the derivation of keys in the Akl-Taylor technique. Identity (1) states that a subkey relation implies the derivation relation. For example, if an information is concealed using the set of keys $\{\kappa^{11}\}$, then any set of keys that contains $\{\kappa^{11}\}$ (e.g., $\{\kappa^{2\times5}, \kappa^{11}\}$) can also be used to get that information. Here, $\{\kappa^{11}\} \leq \{\kappa^{2\times5}, \kappa^{11}\}$, then the key κ^{11} can be derived from $\{\kappa^{2\times5}, \kappa^{11}\}$. Therefore, users involved in more than one hierarchy can combine their keys by using $+_k$ to reveal some information coming from the two hierarchies.

Identity (2) states that a combination of two keys can be derived from any one of them. This can be seen clearly in the Akl-Taylor technique. For example the key $\{k^{3\times5\times7\times9}\}$ can be derived from $\{k^{3\times9}\}$ or the key $\{k^{5\times7}\}$. This identity can be used in key distribution. Assume that we have a key $\{\kappa^{19}\}$ and we need this key to reveal an information that is encrypted using the key $\{\kappa^{2\times5}\}$. The identity shows that if we conceal the information using the set of keys $\{\kappa^{2\times5}\} *_k \{\kappa^{19}\}$ (which is equal to $\{\kappa^{2\times5\times19}\}$), then both keys can be used to reveal that information.

Identity (3) is suitable to specify the situation when two users are involved in two different hierarchies of keys. In this case, if both users update their set of keys with the same key, this does not affect the derivation of keys between them. Identity (4) specifies the situation when a new key is added to the system. Suppose that two users have the keys $k^{5\times7}$ and $k^{5\times7\times9}$. Assume that a new key k^{11} is added to the system that would enable us to derive the keys $k^{5\times7}$ and $k^{5\times7\times9}$. Combining these two keys with k^{11} such that the first key becomes $k^{5\times7\times11}$ and the second one becomes $k^{5\times7\times9\times11}$ does not affect the derivation relation between them. Identity (5) states that 1_k can be used to obtain any key. The identity key 1_k of the key-structure corresponds to the root key in the Akl-Taylor technique and is equal to κ.

3.2 The Chinese Remainder Technique

Chen-Chung [5] propose a technique based on the Chinese remainder theorem. Below, we present the main steps of their technique. We omit the steps related to the cryptosystem because it does not affect the key assignment scheme. Also, note that r_i is called a key and H_i is called a private information [5] while in this paper we call r_i a private information and H_i a key. We changed their naming to be consistent with the terminology adopted in this paper. However, changing

the terminology has no effect since both r_i and H_i are private. The key H_i is used to get r_j for all $c_j \prec c_i$. Note that $\prec$ is a partial order relation.

We suppose that there are m security classes. We select randomly m pairwise co-prime numbers $n_1 \dots n_m$. We take $N = \prod_{1 \leq i \leq m} n_i$, r_i as the private information of class i, and we take y_i as an integer number such that $(N/n_i) \times y_i \pmod{n_i} = 1$. We take $X_i = (N/n_i) \times y_i$, and $H_i = \sum (r_j \times X_j) \pmod{N}$ such that $c_j \prec c_i$. The security class c_i contains the information n_i and r_i such that n_i is public while r_i is private. Each user at class c_i has a key H_i to get the information r_j of other classes as $r_j = H_i \pmod{n_j}$. We provide the following example to illustrate the above technique.

Assume that there are three security classes c_1, c_2, and c_3 such that $c_3 \prec c_2$ and $c_2 \prec c_1$. In this case, we take $m = 3$. Let $n_1 = 3$, $n_2 = 5$, and $n_3 = 7$. We have $N = 3 \times 5 \times 7 = 105$. Let $r_1 = 1$, $r_2 = 2$, and $r_3 = 3$. We have $y_1 = 2$ where $35 \times y_1 \pmod 3 = 1$, $y_2 = 1$ where $21 \times y_2 \pmod 5 = 1$, and $y_3 = 1$ where $15 \times y_3 \pmod 7 = 1$. Also, we have $X_1 = 70$, $X_2 = 21$, and $X_3 = 15$. Finally, we have $H_1 = 70 + 42 + 45 = 157$ and $H_2 = 42 + 45 = 87$.

It can be easily verified that the information r_2 can be obtained from H_1 as $157 \pmod 5 = 2$. The value of r_3 can be obtained from H_1 as $157 \pmod 7 = 3$, and r_1 can be obtained from H_1 as $157 \pmod 3 = 1$.

To add a new security class c_{m+1}, one needs to update all the values of H_i where $c_i \prec c_{m+1}$, and generate the public value of c_{m+1} and its secret information as follows: (1) Select a new prime number n_{m+1}; (2) Let $N = N \times n_{m+1}$; (3) Let $r_{(m+1)}$ be the private information of class $m + 1$; (4) Let y_{m+i} be an integer number such that $(N/n_{m+1}) \times y_{m+1} \pmod{n_{m+1}} = 1$; (5) Let $X_{m+1} = (N/n_{m+1}) \times y_{m+1}$; (6) Update H_i of c_i for $c_{m+1} \prec c_i$ as $H_i = H_i + r_{m+1} \times X_{m+1} \pmod N$; (7) Let $H_{m+1} = \sum (r_j \times X_j) \pmod N$ such that $c_j \prec c_{m+1}$.

Continuing with our example, assume that we add a new class such that that $c_3 \prec c_4$ and $c_4 \prec c_1$. Let $n_{m+1} = 11$. The poset of the scheme is given by Figure 2. Therefore, the new value of N is 1155. Let $r_4 = 4$, we have $y_4 = 2$ as $105 \times y_4 \pmod{11} = 1$. Also, we have $X_4 = 105 \times 2 = 210$. The new value of H_1 becomes $H_1 = 157 + (4 \times 201) = 961$ and the value of H_4 is $45 + 804 = 849$. This technique is not correct as H_1 cannot be used anymore to obtain the values r_2, r_3, or even r_4. For example, $961 \pmod 5 = 1$ while the value of r_2 is 2.

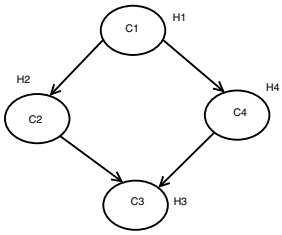


Fig. 2. An example of a hierarchy of classes

Specifying the Chen-Chung Technique
To formally specify the technique of Chen-Chung using our generic algebraic model, we take m security classes where a confidential information r_i is associated with a class c_i. Also, we take m pairwise co-prime numbers $n_1 \cdots n_m$ and $N = \prod_{1 \leq i \leq m} n_i$. Let y_i be an integer number such that $(N/n_i) \times y_i \pmod{n_i} = 1$ and $X_i = (N/n_i) \times y_i$. The key H_i assigned to class c_i is $H_i = \sum((r_j \times X_j) \pmod{N})$ for all $c_j \prec c_i$. It can be easily verified that $H_i \pmod{n_j} = 0$ for $i \neq j$ and $H_i \pmod{n_i} = r_i$. Let the set $S_{i,j} = \{F \mid F = \sum_{i \leq x \leq j} H_x\}$ contains all possible summations of the keys H_i constructed for the m security classes. We define a function *rep* similar to the one defined in the previous subsection that takes a key and returns its set representation.

$$\begin{aligned} &\text{rep} : F \rightarrow \mathcal{P}(\mathcal{P}(F)) \\ &\text{rep}(H_1 + H_2 \cdots H_n) \stackrel{\text{def}}{=} \{\{H_1, H_2, \cdots, H_n\}\} \ . \end{aligned}$$

Let $\mathbb{FF} \stackrel{\text{def}}{=} (\mathcal{P}(\mathcal{P}(F)), +_k, *_k, 0, 1)$ be a structure where the elements of S are sets of subsets of F and where $*_k$ and $+_k$ are defined the same as for our representation of the Akl-Taylor technique on Page 69. It is easily checked that with these definitions of $+_k$ and $*_k$ the structure $\mathbb{FF}$ is a model for the key-structure. Therefore, we can define on it the relations $\leq$ as $\subseteq$ and the relation $\sqsubseteq$ as $a \sqsubseteq b \stackrel{\text{def}}{\iff} \exists(c \mid c \in \mathcal{P}(\mathcal{P}(F)) : a \subseteq b *_k c)$. We can define $k_1 \prec_d k_2 \stackrel{\text{def}}{\iff} k_2 \sqsubseteq k_1$. In this case $\prec_d$ is the dual of $\sqsubseteq$. We can read this relation as every element of the key k_2 can derive all the confidential information r_i which can be derived from some elements of the key k_1.

An Amended Version of to the Chen-Chung Technique
We showed at the beginning of this section that handling the addition of a new class is not correct in the original technique of Chen-Chung. We noticed that the new set of keys cannot be formulated as $S_{i,j}$ and the property of keys $H_i \pmod{n_j} = 0$ for $i \neq j$ and $H_i \pmod{n_i} = r_i$ that is necessary for the correctness of the technique does not hold. The reason for that is the use of two different values of N where the value of N has been changed to $N \times n_{m+i}$ during the addition of a new class.

To correct this method, we should have the value of N fixed and not changed during the addition of a new class. Since the value of N depends on the number of security classes m, we can have m to be the *maximum* number of classes that can be created. In this case, we do not need to change the value of N when a new class is added. We illustrate our idea through the example given previously in this section. Assume that the total number of classes that can be added is 4. Assume that $n_i \in \{3, 5, 7, 11\}$. Therefore, $N = 1155$. Assume the initial scheme contains only three classes where $r_1 = 1$, $r_2 = 2$, and $r_3 = 3$. Therefore, $y_1 = 1$, $y_2 = 1$, $y_3 = 2$, $X_1 = 385$, $X_2 = 231$, $X_3 = 330$. Then, $H_1 = 1837$, $H_2 = 1452$, and $H_3 = 990$. Assume a new class is added where $r_4 = 4$. Now $y_4 = 2$ and $X_2 = 210$. The new $H_4 = 1830$ and the new $H_1 = 1837 + 840 = 2677$.

Algebraic Analysis Based on Proposition 1
Proposition 1 shows some properties of the derivation relation. In this section, we show their relationship to the Chen-Chung technique. Identity (1) shows that if k_1 is a subkey of k_2, then there is some information that can be revealed from k_2 which can also be revealed from k_1. A key k_1 is a subkey of k_2 in this context if it is a subset of k_2.

Identity(2) shows that any information that can be revealed from a key can also be revealed from its $*_k$ combination with another key. In the above example, we have $c_3 \prec c_2$. Therefore, the key $H_2 *_k H_3$, which is translated in our current model to $H_2 + H_3 \pmod N$, is given to the user at class c_2 to reveal all the information that can be revealed by the classes c_2 and c_3. Identity (3) states that if k_1 can reveal information that can be revealed from k_2, then extending both keys would not affect the confidentiality of the information.

Identity(4) covers the case where we have $c_i \prec c_j$ and a new class c_k is added such that $c_k \prec c_i$. For the above example, if we add a new class c_4 such that $c_4 \prec c_1$, the new value of H_1 becomes $H_1 = H_1 *_k H_4$ and the new value of H_2 becomes $H_2 = H_2 *_k H_4$. Identity (4) says that this combination would not affect the ability of H_1 to derive the information that can be derived using H_2. Identity (5) shows that any information that can be derived from the 1_k of the proposed model can be derived from any key.

3.3 Verification

Our key assignment scheme allows us to specify systems that manage key distribution and verify some properties such as the ability of a user to get an information intended for a higher class, or the ability of using several keys to reveal an information that can be revealed by using another key.

Example 2. We are specifying the key assignment scheme for a hospital that contains part-time, over-night, and full-time nurses or doctors. We take the following as the keys distributed to the classes of users. Part-time nurses class c_{pn} gets a key that is constructed using k_1, k_2, and k_4 and is equal to $k_1 *_k k_2 *_k k_4$. Over-night nurses class c_{nn} gets a key that is constructed using k_1, k_3, and k_4 and is equal to $k_1 *_k k_3 *_k k_4$. Full-time nurses class c_{fn} gets a key that is constructed using k_1 and k_4 and is equal to $k_1 *_k k_4$. Part-time doctors class c_{pd} gets a key that is constructed using k_2 and k_4 and is equal to $k_2 *_k k_4$. Over-night doctors class c_{nd} gets a key that is constructed using k_3 and k_4 and is equal to $k_3 *_k k_4$. Full-time doctors class c_{fd} gets the key k_4. Assume that our hospital system should satisfy the following confidentiality properties.

Property 1: A doctor should be able to reveal the information that can be revealed through the key of a nurse that is in the same class. For instance, a part-time doctor can get the information of a part-time nurse. We formally articulate this property as follows: $(k_1 *_k k_4 \prec_d k_4) \wedge (k_1 *_k k_3 *_k k_4 \prec_d k_3 *_k k_4) \wedge (k_1 *_k k_2 *_k k_4 \prec_d k_2 *_k k_4)$.

Property 2: A full time nurse should be able to reveal the information that can be revealed through the key of part-time and over-night nurses. $(k_1 *_k k_3 *_k k_4 +_k k_1 *_k k_2 *_k k_4) \prec_d (k_1 *_k k_4)$.

Property 3: There is no key relationship between part-time and over-night nurses (i.e., user-secure for these two keys). $\neg(k_1 *_k k_2 *_k k_4 \prec_d k_1 *_k k_3 *_k k_4) \wedge \neg(k_1 *_k k_3 *_k k_4 \prec_d k_1 *_k k_2 *_k k_4)$.

Property 4: Part-time and over-night nurses cannot use their keys together to reveal the information that can be revealed through the key of a full-time nurse $\neg(k_1 *_k k_4 \prec_d (k_1 *_k k_3 *_k k_4 +_k k_1 *_k k_2 *_k k_4))$.

In this example we assume that $k_1 \prec_d k_2 \iff k_1 \sqsubseteq k_2$. We verify the above properties by using the key-structure given in Definition 1 with the aid of Prover9 [12], which is an automated theorem prover for proving propositions in first-order and equational logic. We formalise all the axioms of the key-structure. Then, we use Prover9 to verify each property.

Property 1 and Property 2 focus on the correctness of the method. Prover9 established the proof of these properties. Property 3 and Property 4 are harder to prove. Prover9 is not able to prove them. A thorough analysis reveals that we need additional properties on the keys k_1, k_2, k_3, and k_4. These properties/assumptions are ignored in some methods which results in breaches of security. Some of these assumptions are obvious but they should be analysed carefully at the implementation level, otherwise the system can be vulnerable to different attacks. One of the assumptions is that the elementary keys (k_1, k_2, k_3, and k_4) are independent. Another assumption is that combining two keys does not give 0_k or 1_k. Also, we should assume that the property $k_1 \sqsubseteq k_1 *_k k_2$ does not hold. Therefore, the complete specification of the key assignment properties that enables us to prove the Properties 1–4 would be the axioms of the key-structure in addition to the following axioms:

1. $\forall(i,j \mid 1 \leq i \leq 4 \wedge 1 \leq j \leq 4 \wedge i \neq j : \neg(k_i \sqsubseteq k_j))$
2. $\forall(i,j \mid 1 \leq i \leq 4 \wedge 1 \leq j \leq 4 \wedge i \neq j : k_i *_k k_j \neq 0)$
3. $\forall(i,j \mid 1 \leq i \leq 4 \wedge 1 \leq j \leq 4 \wedge i \neq j : k_i *_k k_j \neq 1)$
4. $\forall(i,j \mid 1 \leq i \leq 4 \wedge 1 \leq j \leq 4 \wedge i \neq j : \neg(k_i \sqsubseteq k_i *_k k_j))$

4 Literature Review

One of the widely known techniques proposed for key assignment by deriving one key from another key is the technique introduced by Akl-Taylor [1]. Other techniques are proposed to improve it. For instance, in [11], MacKinnon et al. show that the key generation in the Akl-Taylor technique becomes inefficient in large systems and provide an improved algorithm. In [7], Harn and Lin propose a bottom-up design in key derivation instead of top-down as in the Akl-Taylor technique to improve the storage size of the public data.

In [9], Kuo et al. present a technique based the Chinese remainder theorem (e.g., [17]) and the Rabin cryptosystem [13]. Their technique allows handling the dynamic access control problem such as adding and deleting security classes. Based on the technique of Kuo et al. [9], Chen-Chung [5] present an improved technique that reduces the computation time and the storage size. However, Zhong and Lin [18] show that the part related to adding a new class is incorrect.

Chien and Jan [4] propose a technique that does not need public key cryptography to decrease the cost of computation. Their technique is based on hash functions. In [2], Atallah et al. propose an efficient technique in terms of space and computation that supports an arbitrary graph based also on hash functions. Atallah et al. [2] provide an improvement over Chien and Jan's technique when a security class is deleted. Other techniques such as those of Liaw et al. [10] and Sandhu [16] limit their structure into a tree to increase the efficiency of computation and storage. Crampton et al. [6] present and discuss several generic schemes for key assignment and compare their relative merits.

5 Discussion and Conclusion

In this paper, we applied the key-structure of the framework presented in [14,15] to the problem of key assignment. We present a generic model for key assignment schemes. This model does not depend on a specific crypto-system. It allows us to articulate security properties among security classes to key assignment schemes, then to prove whether the adopted scheme satisfies these properties. The proof is done in an algebraic calculational way that can be easily automated using a theorem prover such as Prover9.

To the best of our knowledge, the only work that classifies key assignment schemes is the work of Crampton et al. [6]. In [6], the authors introduce five generic schemes and claim that every existing technique is an instance of one of their schemes. We can represent all of these schemes within our mathematical structures. We summarize these schemes and relate them to our structures. The first scheme is the *trivial key assignment scheme*. This scheme allows each user the handling of a set of keys. Each key can be used to reveal a secret. The construction of keys can be represented as $+_k$. The second scheme is the *trivial key encrypting key assignment scheme*. This scheme is based on encrypting the required key for revealing the secret using the user key. This scheme does not involve keys only or their properties but takes secrets and ciphers into consideration. This scheme can be specified by using the key-structure together with the secrets and ciphers structures presented in [14,15]. Similarly, the third is a *direct key encrypting key assignment scheme* where the user keys are encrypted. The fourth scheme is an *iterative key encrypting key assignment scheme* and is similar to the third scheme but decryption is in an iterative way. Finally, the fifth scheme is based on deriving a key from another one as in the Akl-Taylor technique. We already illustrated how it can be handled within our proposed model.

To the best of our knowledge, this is the first work that analyzes formally key derivation at this abstract level and in a generic way. We intend as a future work to investigate other key assignment schemes to assess their strengths and weaknesses.

References

1. Akl, S., Taylor, P.: Cryptographic solution to a problem of access control in a hierarchy. ACM Transaction on Computer Systems 1(3), 239–248 (1983)
2. Atallah, M.J., Blanton, M., Fazio, N., Frikken, K.B.: Dynamic and efficient key management for access hierarchies. ACM Transactions on Information and System Security 12(3), 1–43 (2009)
3. Boyd, C.: Some Applications of Multiple Key Ciphers. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 455–467. Springer, Heidelberg (1988)
4. Chien, H.-Y., Jan, J.-K.: New hierarchical assignment without public key cryptography. Computers & Security 22(6), 523–526 (2003)
5. Chen, T.-S., Chung, Y.-F.: Hierarchical access control based on Chinese remainder theorem and symmetric algorithm. Computers & Security 21(6), 565–570 (2002)
6. Crampton, J., Martin, K., Wild, P.: On key assignment for hierarchical access control. In: Proceedings of the 19th IEEE workshop on Computer Security Foundations (CSFW 2006), Venice, Italy, pp. 98–111. IEEE Computer Society (2006)
7. Harn, L., Lin, H.-Y.: A cryptographic key generation scheme for multilevel data security. Computer Security 9(6), 539–546 (1990)
8. Höfner, P., Khedri, R., Möller, B.: Feature Algebra. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 300–315. Springer, Heidelberg (2006)
9. Kuo, F.H., Shen, V.R.L., Chen, T.S., Lai, F.: Cryptographic key assignment scheme for dynamic access control in a user hierarchy. IEEE Proceedings Computers and Digital Techniques 146(5), 235–240 (1999)
10. Liaw, H.T., Wang, S.J., Lei, C.L.: A dynamic cryptographic key assignment scheme in a tree structure. Computers & Mathematics with Applications 25(6), 109–114 (1993)
11. MacKinnon, S.J., Taylor, P.D., Meijer, H., Akl, S.G.: An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. IEEE Transactions on Computers 34(9), 797–802 (1985)
12. McCune, W.: Prover9 and Mace4, `http://www.cs.unm.edu/~mccune/prover9/`
13. Rabin, M.: Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science (1979)
14. Sabri, K.E.: Algebraic Framework for the Verification of Confidentiality Properties. PhD thesis, McMaster University (2010)
15. Sabri, K.E., Khedri, R.: Algebraic framework for the specification and analysis of cryptographic-key distribution. Fundamenta Informaticae 112(4), 305–335 (2011)
16. Sandhu, R.S.: On some cryptographic solutions for access control in a tree hierarchy. In: ACM 1987: Proceedings of the 1987 Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow, pp. 405–410. IEEE Computer Society Press, Los Alamitos (1987)
17. Yan, S.Y.: Number theory for computing. Springer (2002)
18. Zhong, S., Lin, T.: A comment on the chen-chung scheme for hierarchical access control. Computers & Security 22(5), 450–452 (2003)

Message Transmission and Key Establishment: Conditions for Equality of Weak and Strong Capacities

Hadi Ahmadi and Reihaneh Safavi-Naini

Department of Computer Science, University of Calgary
{hahmadi,rei}@ucalgary.ca

Abstract. Secure communication using noisy resources has been first studied in the contexts of secure message transmission (SMT) by Wyner as well as Csiszár-and-Körner, and secret key establishment (SKE) by Ahlswede-and-Csiszár as well as Maurer. The work defines secrecy (resp. secret-key (SK)) capacity as the highest achievable rate of secure transmission (resp. key establishment). Maurer and Wolf later focused on SKE and noticed that the secrecy requirement in the SK capacity definition was weak as it required only the "ratio" between the adversary's information and the key length to be negligible. They suggested a stronger definition of the SK capacity by requiring absolute information leakage to be negligible. They provided an interesting proof for the equality of weak and strong SK capacities in the above scenarios (setups).

Followup work has since studied several setups for SKE by considering the weak SK capacity without discussing whether the results also hold for the strong definition. In this paper, we pose the question whether the equality of weak and strong SK capacities can be derived in general for all discrete memoryless communication setups. We also extend this study to message transmission and investigate the equality of weak and strong secrecy capacities. For SKE, we show that weak and strong SK capacities are equal for any setup that allows reliable transmission in any direction. For SMT, the secrecy capacities are equal when the setup allows the sender to use randomness. We furthermore provide trivial counterexamples that show these sufficient conditions are not always necessary for the equality of the capacities. Whether the conditions can be removed or relaxed by tight (necessary and sufficient) conditions remains an interesting question for future.

Keywords. Secret key capacity, secrecy capacity, information reconciliation, privacy amplification.

1 Introduction

Wyner [21] and subsequently Csiszár and Körner [9] pioneered the study of secure message transmission (SMT) over noisy wiretap channels. Their study shows when the wiretapper's channel is noisier than the main channel, the sender can send arbitrarily long messages securely by using long enough codes over the

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 78–94, 2013.

wiretap channel. Since communication resources are generally expensive, it is preferred to design "optimal" codes which send more message bits securely by using the channel as few number of times as possible. This leads to the definition of the secrecy capacity, i.e., the highest information rate (in bits per channel use) that can be sent in a secure and reliable manner. The work in [9,21] proved how to obtain this capacity for a wiretap channel. Maurer [14] considered the above communication scenario for secret key establishment (SKE), where he defined the secret-key (SK) capacity as the highest rate of secret key that can be shared by using the communication resources. For the one-way wiretap channel [9, 21], the SK capacity equals the secrecy capacity as the highest key rate is achieved by sending a random key using a secure message transmission code. Maurer [14] thus revisited the SKE problem assuming a public discussion channel in addition to correlated randomness (e.g., obtained from a wiretap channel). The public discussion channel is a free noiseless channel that can be used unlimitedly by the parties in both directions; however, its content is completely observable by the eavesdropper. Maurer proved that the SK capacity in the new "setup" is strictly higher than that of a single wiretap channel. Similar results were independently derived by Ahlswede and Csiszár [1].

In the definition of the SK capacity in [1, 14], the secrecy condition requires the SKE protocol to leak negligible information about the key; however, the information leakage is measured in "rate" (per key bit on average). This implies that the absolute amount of leakage can be unbounded (as the key length increases), although the leakage rate is close to zero. Maurer and Wolf [15] refer to this definition as the weak SK capacity and suggest strong SK capacity which, rather than leakage rate, requires the SKE protocol to leak "absolutely" negligible information about the key. This raises two questions: (1) is it possible to achieve strongly secret keys for setups in [1, 9, 14, 21], and (2) is this strengthening doable without sacrificing the key rate? The work in [15] gave positive answers to both questions using an elegant construction of strongly-secure SKE protocols from weakly secure ones, which proves the equality of the weak and strong SK capacities for a wiretap channel with/without public discussion.

Thee has been since a lot of research on SKE in various setups, where the SK capacity has been investigated. Most of them however considered weak secrecy without discussing whether the results on the SK capacity also hold when strong secrecy is required (cf. [2–5,11,13,18]). This has once again brought about ambiguity in the relation between weak and strong SK capacities for many communication setups considered thereafter. In addition to the above, there is a similar question about the equality of weak and strong secrecy capacities in secure message transmission. To the best of our knowledge, this latter question has not been addressed by the work in [15] or any work thereafter.

1.1 Motivation

Investigating the equality of weak and strong secrecy capacities (for SMT) is reasonably important since it is different from SKE and it has not been covered by the previous work. One may however ask about the importance of reinvestigating

the equality of capacities in SKE noting that Maurer-and-Wolf's (MW) proof [15] is quite generic to be adapted to other setups. In what follows, we argue that the MW approach does not apply to "all" existing communication setups and needs to be re-examined to see when it stays valid.

The study in [15] shows the equality of weak and strong SK capacities in the following two settings: (1) correlated randomness and public discussion channel (PDC) [1, 14], and (2) one-way discrete memoryless wiretap channel (DMWC). We focus on their proof method (for the second setup because it is more generic and does not rely on PDC as a free noiseless channel resource. The MW approach proceeds in two phases. The first phase shows that the weak SK capacity is equal to the so-called uniform SK capacity (which requires the key distribution to be close to uniform). The second phase shows that it is possible to construct a strongly secure SKE protocol using a uniformly-secure protocol without sacrificing the key rate. This construction mainly consists of four steps: (i) *independent repetition* of the uniformly-secure protocol, (ii) *information reconciliation* with universal hashing, (iii) *privacy amplification* using a seeded-extractor, and (iv) *uniformization*. For information reconciliation, Alice uses the wiretap channel to send error-correction bits. For privacy amplification, she uses her (free) independent source to generate uniformly random bits and sends them to Bob over the wiretap channel, so they both agree on the random seed used in the extractor. By repeating the uniformly-secure protocol sufficiently many times, in step (i), the number of channel uses in steps (ii) and (iii) becomes negligible and the key rate tends toward that of the uniformly-secure protocol.

The MW approach is indeed generic enough to work for other discrete memoryless setups. Revisiting this approach, one can assure that a slight rephrasing of the key rate analysis makes it applicable to most of the current setups. Such setups include the correlated randomness and wiretap channel setup in [13, 18] and a pair of wiretap channels in [2, 3]. Yet, the proof is based on two main assumptions that make it not adaptable to "all" setups.

- There is a channel with positive (reliability) capacity in at least one direction.
- There is a free local source of randomness available to at least one party.

The first assumption is hidden in the proof as for the one-way DMWC, *the secrecy capacity is "equal" to the SK capacity:* When the weak SK capacity is positive, the channel is capable of (secure hence) reliable transmission needed for information reconciliation. The above property does not hold for many communication scenarios such as [2,4,5,13,18]. More crucially, there are instances of two-way wiretap channels [4] where, in spite of positive SK capacity, it is impossible to have even a single bit of reliable transmission. This implies that reliable transmission is not generally an implicit capability of all setups with positive SK capacity and hence is required to be checked separately in each case.

Similarly, the second assumption does not hold in all scenarios, e.g., when local randomness is not free. We note that the freeness of local randomness is not the main issue. The MW approach requires generation of uniform randomness of length negligible to the key length; hence, using a local random source with a constant cost does not affect the key rate analysis and so the equality result.

However, the existence of local randomness is crucially required by the MW approach. The secret-key from noise scenario in [5] assumes no local random source in the system and this renders the MW approach inapplicable.

1.2 Our Work

We provide a formal treatment of the problems by studying SMT and SKE in a general *discrete memoryless communication setup*. We define a *setup* as a set of communication resources, i.e., sources and channels, with certain underlying assumptions that model a communication scenario between Alice and Bob in the presence of a passive adversary, Eve. A *discrete memoryless* setup is one in which all channels and sources are discrete-alphabet and memoryless and can be used as many times as required.

The SKE Problem. We investigate the relation between the weak and strong SK capacities in general, by acquiring whether any of the two assumptions by the MW approach can be removed. Fortunately, we can show that the second assumption is not necessary: The equality of weak and strong SK capacities holds regardless of whether or not access to local sources of randomness is granted. We prove this by modifying step (iii) of the MW approach: Our privacy amplification requires a two-source extractor instead of a seeded extractor. The two inputs for the two-source extractor are obtained by two independent instances of steps (i) and (ii); this leads to an interesting consequence that our privacy amplification step does not depend on any resources in the setup, in contrast with that of the MW approach that needs resources to generate random bits as well as to send them to the other party. Our analysis shows that the modified construction still results in a strongly-secure SKE protocol that achieves the weak SK capacity.

We however do not know whether the need for reliable transmission (from Alice to Bob or vise versa) during information reconciliation can be relaxed or removed. This leaves us with the following conclusion.

The weak and strong SK capacities equal in any discrete memoryless setup that allows positive-rate reliable data transmission in at least one direction.

The SMT Problem. We give a construction of a strongly-secure SMT protocol using a weakly-secure SMT protocol. The construction consists of the following steps: The sender (i) *expands* the message using a two-source extractor inverter, (ii) *splits* the message to many independent message pieces and sends each by the weakly-secure protocol, (iii) sends error correction bits for *information reconciliation*, and the receiver (iv) *extracts* the message by the two-source extractor. This construction is strongly-secure and achieves the weak secrecy capacity. The construction needs randomness for the sender in step (ii), and reliable transmission in step (iv) for information reconciliation. It is easy to observe that reliable transmission requirement can be removed since any setup with positive (weak) secrecy capacity allows reliable message transmission too. However, it remains open whether we can remove or relax the need for randomness in step (ii). This is

exactly dual to the SKE problem where the remaining condition is the possibility of reliable transmission. The conclusion is the following.

The weak and strong secrecy capacities equal in any discrete memoryless setup that allows the sender to use randomness.

1.3 Discussion

Setup Assumptions. We consider only "discrete memoryless setups". This family of setups is large enough to capture a great number of studies on SKE and SMT. Besides, our results also hold for setups with continuous alphabet resources (e.g., Gaussian sources and channels [20]) as well as state-dependent channels [16] with independently and identically distributed (i.i.d.) state sequence.

Message Transmission vs. Key Agreement. The results on the equality of weak and strong capacities in SMT and SKE are quite related, but not identical. In SKE, the parties aim to share a secret that is not necessarily known before the protocol. In SMT however, the sender aims to deliver an existing message. If a protocol leaks information about the message, it is impossible compensate this leakage as the message cannot be changed. This makes us use an extractor inverse function to expand a message before sending it.

Equality Requirements: Necessity vs. Sufficiency. The conditions that we derive for the equality of the weak and strong capacities are not always necessary. In Section 6, we discus special cases to support this. It remains yet an interesting problem to find whether we can remove the conditions from the claims or there are tighter (necessary and sufficient) conditions for the equality.

1.4 Notation

We use calligraphic letters ($\mathcal{X}$), uppercase letters (X), and lowercase letters (x) to denote finite alphabets, random variables, and their realizations over sets, respectively. We denote the length of sequences by using superscripts (x^n). For two random variables X and Y over alphabets $\mathcal{X}$ and $\mathcal{Y}$, we denote by P_X (or P_Y), $P_{X,Y}$ and $P_{Y|X}$ the marginal, joint, and conditional probability distributions, respectively, and by $H(X,Y)$ and $H(Y|X)$ the joint and conditional (Shannon) entropies, respectively. We furthermore use $H_\infty(X) = -\log(\max_x P_X(x))$ to denote the min-entropy of X and use $\| X - Y \|_s$ to denote the statistical (variational) distance [8] between the two random variables X and Y.

2 Model, Definitions, and Results

2.1 Discrete Memoryless Communication Setup

Consider a communication scenario with legitimate parties Alice and Bob who want to achieve a security goal in the presence of a passive adversary, Eve. Depending on the communication model, Alice, Bob, and Eve may have access to

specific types of sources and channels. *We refer to a communication scenario that specifies a set of resources, i.e., sources and channels, available to the parties by a setup.* The performance of a message transmission (or key agreement) protocol over a setup is measured in terms of rate, that is the number of message (or key) bits divided by the cost imposed to the setup by the protocol. This overall cost is the sum of those costs imposed to each resource. The cost for a resource is generally a function of how many times the resource is used as well as a *per-use cost* value (the cost of using the resource only once). In this work, We particularly consider *discrete memoryless setups*, wherein resources have the following properties:

- Each resource is discrete-alphabet and memoryless, i.e., using the resource independently many times results in independent outputs.
- Each resource can be used for as many, n, times as required: the total cost for such a resource is obtained by multiplying its per-use cost by n.

The above conditions let us give abstract definitions for general channel/source resources. We define a channel as an alphabet tuple $(\mathcal{X}_A, \mathcal{X}_B, \mathcal{Y}_A, \mathcal{Y}_B, \mathcal{Y}_E)$, a channel probability distribution $P_{Y_A,Y_B,Y_E|X_A,X_B}$ over the alphabets, and a per-use cost value. The alphabets respectively denote the channel inputs from Alice and Bob as well as the channel outputs to Alice, Bob, and Eve. This channel model is an extension of Shannon's two-way discrete memoryless channel [19] to when it leaks to a wiretapper [4]. Notice that the model captures any discrete-alphabet, memoryless channel resource in the literature (cf. [1,2,9,11,14,21]).

We furthermore define a source as an alphabet triple $(\mathcal{X}_A, \mathcal{X}_B, \mathcal{X}_E)$, a probability distribution P_{Y_A,Y_B,X_E}, and a per-use cost. This definition lets us capture any independent (local) or correlated source(s) of randomness [1,13,14,18] with arbitrary cost value. We note some communication scenarios, such as [5], may not assume any such sources in their setup. The following gives a formal definition for cost of a protocol using a setup.

Definition 1 (Cost). *Let $\mathfrak{S}$ be a discrete memoryless setup that consists of m resources $\mathfrak{R}_1, \mathfrak{R}_2, \ldots, \mathfrak{R}_m$ with per-use costs $c_1, c_2, \ldots, c_m$, respectively. Let Π be a protocol over the setup that uses the above resources for $n_1, n_2, \ldots, n_m$ number of times, respectively. The cost of executing Π over $\mathfrak{S}$ is defined as*

$$Cost_{\Pi}^{\mathfrak{S}} = \sum_{i=1}^{m} c_i n_i. \tag{1}$$

The above definition of cost becomes much simpler considering the currently existing cases. Many communication setups in the literature [1,4,9,14,21] include only one costly resource with the per-use cost of 1 (the rest of the resources are free), implying that the total cost equals to the number of times the resource is used. There are also examples of setups with two costly resources, such as a pair of DMWCs [2,3] and DMWC with multiple-sources [13], where the total cost is obtained by adding the number of times each resource is used.

2.2 Message Transmission and Key Establishment

In this part, we describe reliable message transmission (RMT), secure message transmission (SMT), and secret key establishment (SKE) over a discrete memoryless setup. In RMT, either Alice or Bob wants to send a message reliably to the other party; however, the message is not required to be kept secure from Eve. We use *forward (or backward) direction* to mention the direction of message transmission from Alice to Bob (or vice versa). An RMT protocol may proceed in multiple rounds possibly to achieve higher transmission rates.

Definition 2 (Reliability capacity). *For real constants $R \geq 0$ and $\delta > 0$, an RMT protocol, Π, in a discrete memoryless setup, $\mathfrak{S}$, is called (R, δ)-reliable if it allows the sender to send a K-bit uniform message M (for large enough K), where the receiver receives $\hat{M}$ and the following holds*

$$\textit{reliability:} \quad \Pr(M = \hat{M}) \geq 1 - \delta. \tag{2a}$$

$$\textit{rate:} \quad \frac{K}{Cost_{\Pi}^{\mathfrak{S}}} \geq R - \delta. \tag{2b}$$

The forward (resp. backward) reliability capacity of the setup $\mathfrak{S}$ is the largest $R \geq 0$ such that, for any arbitrarily small $\delta > 0$, there exists an (R, δ)-reliable forward (resp. backward) RMT protocol. The maximum reliability capacity $C_m^{\mathfrak{S}}$ is the maximum of the forward and the backward reliability capacities.

Secure message transmission is an extension of the above when the message is required to remain secure given Eve's view of the communication.

Definition 3 (Secrecy capacity). *For real constants $R_s \geq 0$ and $\delta > 0$, a SMT protocol, Π, as in Definition 2 is called (R_s, δ)-weakly, respectively, -strongly secure if in addition to (2), the following holds*

$$\textit{weak secrecy:} \quad H(M|View_E) \geq K(1 - \delta), \tag{3}$$

respectively,

$$\textit{strong secrecy:} \quad H(M|View_E) \geq K - \delta, \tag{4}$$

where $View_E$ is Eve's view of the communication. The weak/strong forward/backward secrecy capacity of the setup $\mathfrak{S}$, respectively denoted by $C_{wfs}^{\mathfrak{S}}$, $C_{wbs}^{\mathfrak{S}}$, $C_{sfs}^{\mathfrak{S}}$, and $C_{sbs}^{\mathfrak{S}}$, is the largest $R_s \geq 0$ such that, for any arbitrarily small $\delta > 0$, there exists an (R, δ)-weakly/strongly forward/backward SMT protocol.

In secret key establishment, Alice and Bob use the resources to generate random variables S_A and S_B over a set $\mathcal{S}$, as the key estimates, in a reliable and secure way: the reliability means that $S_A = S_B$ with high probability and the security requires Eve's view $View_E$ not to reveal information about S_A (or S_B).

Definition 4 (Weak SK capacity). *For real constants $R_s \geq 0$ and $\delta > 0$, a SKE protocol, Π, in a discrete memoryless setup, $\mathfrak{S}$, is called (R_s, δ)-weakly-secure if it results in S_A, S_B, and $View_E$ as defined above such that it holds*

$$\textit{reliability:} \quad \Pr(S_A = S_B) \geq 1 - \delta. \tag{5a}$$

$$\textit{weak secrecy:} \quad H(S_A|View_E) \geq H(S_A)(1 - \delta), \tag{5b}$$

$$\textit{randomness:} \quad H(S_A) \geq Cost_{\Pi}^{\mathfrak{S}}(R_s - \delta), \tag{5c}$$

The weak secret key (SK) capacity, $C_{wsk}^{\mathfrak{S}}$, is the largest R_s such that, for any arbitrarily small $\delta > 0$, there exists an (R_s, δ)-weakly-secure SKE protocol.

Definition 4 has two weaknesses. The first is the key randomness: (5c) does not require the key to be even close to uniform. The second weakness is the secrecy: (5b) requires Eve's uncertainty about the key to be arbitrarily small, only in rate. Uniformity is partially addressed in Definition 5.

Definition 5 (Uniform SK capacity). *The SKE protocol Π in Definition 4 is called (R_s, δ)-uniformly-secure if in addition to (5a)-(5c) it holds that*

$$\textit{uniformity:} \quad H(S_A) \geq \log|\mathcal{S}| - \delta Cost_{\Pi}^{\mathfrak{S}}. \tag{6}$$

The uniform SK capacity, $C_{usk}^{\mathfrak{S}}$, is the largest R_s such that, for any arbitrarily small $\delta > 0$, there exists an (R_s, δ)-uniformly-secure SKE protocol.

We define strongly-secure protocols that satisfy stronger uniformity and secrecy conditions: (i) the key $S_A \in \mathcal{S}$ is perfectly uniform and (ii) the secrecy condition is on Eve's absolute uncertainty, rather than the rate.

Definition 6 (Strong SK capacity). *A SKE protocol Π, as in Definition 4, is called (R_s, δ)-strongly-secure the following holds*

$$\textit{reliability:} \quad \Pr(S_A = S_B) \geq 1 - \delta. \tag{7a}$$

$$\textit{strong secrecy:} \quad H(S_A|View_E) \geq H(S) - \delta, \tag{7b}$$

$$\textit{strong uniformity:} \quad H(S_A) = \log|\mathcal{S}| \geq (R_s - \delta)Cost_{\Pi}^{\mathfrak{S}}. \tag{7c}$$

The strong SK capacity, $C_{ssk}^{\mathfrak{S}}$, is the largest R_s such that, for any arbitrarily small $\delta > 0$, there exists an (R_s, δ)-strongly-secure SKE protocol.

2.3 Main Results

Definition 3 trivially implies that strong secrecy in message transmission implies weak secrecy. Similarly, one can observe from Definitions 4-6 that strong security in SKE implies uniform security, which itself implies weak security.

Proposition 1. *For any discrete memoryless setup $\mathfrak{S}$, an (R_s, δ)-strongly-secure SMT protocol with at least one message bit is (R_s, δ)-weakly-secure, implying*

$$C^{\mathfrak{S}}_{sfs} \leq C^{\mathfrak{S}}_{wfs} \quad \textit{and} \quad C^{\mathfrak{S}}_{sbs} \leq C^{\mathfrak{S}}_{wbs}.$$

Proposition 2. *For any discrete memoryless setup $\mathfrak{S}$, any (R_s, δ)-uniformly-secure SKE protocol is (R_s, δ)-weakly-secure and any (R_s, δ)-strongly-secure SKE protocol (with at least one key bit) is (R_s, δ)-uniformly-secure. This implies that*

$$C^{\mathfrak{S}}_{ssk} \leq C^{\mathfrak{S}}_{usk} \leq C^{\mathfrak{S}}_{wsk}.$$

In the rest of the paper, we acquire conditions that allow us to write the above inequalities in the opposite direction. The ultimate results of this paper are the following two theorems.

Theorem 1. *For any discrete memoryless setup $\mathfrak{S}$, the weak and strong SK capacities are equal if the setup allows for reliable transmission in at least one direction, i.e., $C^{\mathfrak{S}}_m > 0$.*

Theorem 2. *For any discrete memoryless setup $\mathfrak{S}$, the weak and strong (forward or backward) secrecy capacities are equal if the setup allows the sender to use randomness.*

3 Preliminaries

We define the main concepts and primitives used in our proofs. We start by typical sequences whose occurrence probability is close to a typical probability.

Definition 7. [8, Chapter 3] *For $\epsilon > 0$, integer $n > 0$, and distribution P_X over the set $\mathcal{X}$, the sequence x^n is called* ϵ-typical *with respect to P_X if*

$$|nH(X) + \log P(x^n)| \leq n\epsilon, \qquad \textit{where} \qquad P(x^n) = \prod_{i=1}^{n} P_X(x_i).$$

Consider a source that generates independently and identically distributed (i.i.d.) symbols according to a certain distribution. Lemma 1 shows that the source output is, with high probability, a typical sequence as its length increases.

Lemma 1. [8, Chapter 3] *Let X^n be a random sequence of length n which is drawn i.i.d. according to the probability distribution P_X. For any $\epsilon > 0$, for sufficiently large $n > 0$, the sequence X^n is, with probability at least $1 - \epsilon$, an ϵ-typical sequence with respect to P_X.*

The following, which is a straightforward simplification of [15, Lemma 6], shows a somehow similar result about the min-entropy of i.i.d. sequences.

Lemma 2. *For any joint distribution P_{XZ}, let X^n and Y^n be drawn i.i.d. according to P_{XZ}. For any $\epsilon > 0$, for sufficiently large n, there exists an event $\mathcal{E}$ such that $\Pr(\mathcal{E}) \geq 1 - \epsilon/n$, and furthermore, for all $z \in \mathcal{Z}^n$,*

$$H_\infty(X^n | Z^n = z, \mathcal{E}) \geq n\left(H(X|Y) - \epsilon\right).$$

Strengthening the notion of security from weak to strong includes two fundamental concepts, namely information reconciliation and privacy amplification. Information reconciliation can be attained by sending error-correction information from a universal family of hash functions.

Definition 8. [6] *A family $\mathcal{H}$ of hash functions $h : \mathcal{X} \to \mathcal{Y}$ is called universal if, for any x_1 and x_2 in $\mathcal{X}$, the equality $h(x_1) = h(x_2)$ happens with probability $1/|\mathcal{Y}|$ when $h(.)$ is chosen uniformly at random from $\mathcal{H}$.*

Lemma 3. [15] *Let X^n and Y^n be random sequences of length n drawn i.i.d. according to the joint probability distribution $P_{X,Y}$. For any $\epsilon > 0$, for sufficiently large $n > 0$, $L = (1+\epsilon)nH(X|Y)$, and any universal family $\mathcal{H}$ of functions from $\mathcal{X}^n$ to $\{0,1\}^L$, there exists a function $h \in \mathcal{H}$ such that X^n can be decoded from Y^n and $h(X^n)$ with error probability at most ϵ.*

Privacy amplification is the task of compressing a weakly-secure key to strongly-secure key. This purpose can be achieved by a (seeded) extractor.

Definition 9. [17] *For positive integer k and positive ϵ, a function $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^r$ is called a (k,ϵ)-extractor if, for any random variable $X \in \{0,1\}^n$ with $H_\infty(X) \geq k$, it holds $\| Ext(X, U^d) - U^r \|_s \leq \epsilon$, where U^l is an independent uniform distribution over $\{0,1\}^l$. The function is called a (k,ϵ)-strong-extractor if $\| [U^d, Ext(X, U^d)] - U^{d+r} \|_s \leq \epsilon$.*

Definition 9 relies on the existence of a uniform random seed. Not all discrete memoryless setups provide users with uniform randomness. Hence, we apply two-source extractors that extract strongly-secure keys from two independent weakly-secure keys. We also recall a recent result on their constructions.

Definition 10. [7] *For positive integers k_1, k_2 and positive ϵ, a function $TExt : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^r$ is called a (k_1, k_2, ϵ)-two-source-extractor if, for any two independent random variables $X_1 \in \{0,1\}^{n_1}$ with $H_\infty(X_1) \geq k_1$ and $X_2 \in \{0,1\}^{n_2}$ with $H_\infty(X_2) \geq k_2$, it holds $\| TExt(X_1, X_2) - U^r \|_s \leq \epsilon$.*

Lemma 4. [12] *For any choice of parameters $n > 0$, $0 < k_1 \leq n$, $0 < k_2 \leq n$, and $\epsilon > 0$ that satisfy $k_1 + k_2 \geq n + \Omega(\text{polylog}(n/\epsilon))$, there exists an efficient (k_1, k_2, ϵ)-two-source-extractor, $TExt : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^r$, with*

$$r = \max(k_1, k_2) + k_1 + k_2 - n - 4\log(1/\epsilon).$$

To convert a weakly-secure SMT protocol to a strongly-secure one, we use extractor inverters, defined as follows.

Definition 11. *For a two-source extractor $TExt : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \to \{0,1\}^r$, its* inverter *is a deterministic function $Inv : \{0,1\}^\rho \times \{0,1\}^r \to \{0,1\}^{n_1} \times \{0,1\}^{n_2}$ with $\rho = n_1 + n_2 - r$ such that for Rnd uniform over $\{0,1\}^\rho$, the random variable $Inv(Rnd, Y)$ is uniformly selected from the set $\{(X_1, X_2) \in \{0,1\}^{n_1} \times \{0,1\}^{n_2} : TExt(X_1, X_2) = Y\}$.*

4 Proof of Theorem 1: Weak to Strong SK Capacity

Similarly to Maurer and Wolf's approach [15], we divide the proof in two phases: (1) the equality of weak and uniform SK capacities and (2) the equality of uniform and strong capacities.

4.1 Equality of Weak and Uniform SK Capacities

The first phase of the proof exactly follows the MW approach [15, Lemma 5] in constructing, for arbitrarily small $\delta > 0$, an (R_s, δ)-uniformly-secure SKE protocol Π_u by using an (R_s, δ')-weakly-secure protocol Π_w (for suitably small δ'). The protocol Π_u is obtained roughly by repeating the protocol Π_w independently many times to get sequences of i.i.d. secret keys, and accept these sequences only if they are ϵ-typical with respect to the output distribution of Π_w, for suitably small $\epsilon > 0$ (determined from δ). The protocol Π_u is proved to be uniformly-secure and to have the same rate as that of Π_w. This proof does not depend on any communication resources (sources/channels) in addition to those required repeatedly for executing the weakly-secure protocol Π_w.

Lemma 5. [15, Lemma 5] *For any discrete memoryless setup $\mathfrak{S}$, the weak and uniform SK capacities are equal, i.e., $C^{\mathfrak{S}}_{wsk} = C^{\mathfrak{S}}_{usk}$.*

4.2 Equality of Uniform and Strong SK Capacities

The equality means for any $\delta > 0$, there exists a $(C^{\mathfrak{S}}_{usk}, \delta)$-strongly-secure protocol Π_s. The MW approach [15] to show this for setups in [9,14,21] constructs Π_s by using a $(C^{\mathfrak{S}}_{usk}, \delta')$-uniformly-secure protocol Π_u for suitably small $\delta' > 0$. The construction is in four steps: (i) independent repetition of Π_u, (ii) information reconciliation, (iii) privacy amplification, and (iv) uniformization.

Our method of constructing Π_s is similar to the above except for step (iii), privacy amplification, where we use a (deterministic) two-source extractor in place of the seeded extractor. We denote the keys returned by the $(C^{\mathfrak{S}}_{usk}, \delta')$-uniformly-secure protocol Π_u by $S_{uA} \in \mathcal{S}_u$ for Alice and $S_{uB} \in \mathcal{S}_u$ for Bob. We also use V_{uE} to denote Eve's view of this protocol. Let $K = \log|\mathcal{S}_u|$, N be a sufficiently integer, and L and r be integers such that $L \leq \delta_1 NK$ and $r \geq 2NK(1-\delta_4)$, for sufficiently small δ_1, δ_4 as mentioned in Appendix A.

Step (i): Independent Repetition. Alice and Bob repeat Π_u over the setup $\mathfrak{S}$ independently $2N$ times. This results in the pairs of independent sequences (S^N_{uA1}, S^N_{uA2}) for Alice, (S^N_{uB1}, S^N_{uB2}) for Bob, and the view (V^N_{uE1}, V^N_{uE2}) for Eve.
* This step requires resources for $2N$ repetition of Π_u that costs $2NCost^{\mathfrak{S}}_{\Pi_u}$.

Step (ii): Information Reconciliation. Either party finds suitable functions h_1 and h_2 from a universal family of hash functions $\mathcal{H} : \mathcal{S}^N_u \to \{0,1\}^L$, applies h_1 and h_2 to their pair of sequences and sends the outputs reliably (not securely) to the other party. This additional information lets the parties come up with equal

pairs of sequences with high probability. For instance, Alice can find functions h_1 and h_2 in $\mathcal{H}$ such that the knowledge of $(h_1(S^N_{uA1}), h_2(S^N_{uA2}))$ together with (S^N_{uB1}, S^N_{uB2}) can be used to obtain (S^N_{uA1}, S^N_{uA2}) with high probability. We denote by (S'^N_{uA1}, S'^N_{uA2}) and (S'^N_{uB1}, S'^N_{uB2}) the sequences owned by Alice and Bob at the end of this step, respectively. We also use (V_{rE1}, V_{rE2}) to denote Eve's view of the information reconciliation step for the two key sequences, respectively.

* This step requires resources for transmission of $2L$ bits reliably in either direction.

Step (iii): Privacy Amplification. Let $Bin : \mathcal{S}_u^N \to \{0,1\}^n$, where $n = \lceil NK \rceil$, be an injective binary mapping function. Alice calculates the n-bit strings $\tilde{S}_{A1} = Bin(S'^N_{uA1})$ and $\tilde{S}_{A2} = Bin(S'^N_{uA2})$, and Bob calculates $\tilde{S}_{B1} = Bin(S'^N_{uB1})$ and $\tilde{S}_{B2} = Bin(S'^N_{uB2})$. Alice and Bob apply the two-source extractor $TExt : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^r$ of Lemma 4 on their n-bit strings to extract strongly-secure keys over the set $\mathcal{S} = \{0,1\}^r$. Alice calculates $\tilde{S}_A = TExt(\tilde{S}_{A1}, \tilde{S}_{A2})$ and Bob calculates $S_B = TExt(\tilde{S}_{B1}, \tilde{S}_{B1})$. Eve's overall view of this protocol is denoted by $View_E = (V^N_{uE1}, V^N_{uE2}, V_{rE1}, V_{rE2})$.

Step (iv): Uniformitarian.. Alice obtains S_A by sending $\tilde{S}_A$ over a probabilistic channel, with uniform distribution $P_{S_A|\tilde{S}_A}$, that minimizes the error probability $\Pr(S_A \neq \tilde{S}_A)$ over all such distributions.

Appendix A shows that Π_s is a $(C^{\mathfrak{S}}_{usk}, \delta)$-strongly-secure protocol for arbitrarily small $\delta > 0$. The requirement for the proof is that the reliability capacity $C^{\mathfrak{S}}_m$ be positive, since the construction requires to send $2L$ information bits reliably in one direction. Combining Lemmas 5 and 6 completes the proof of Theorem 1.

Lemma 6. *Let the maximum reliability capacity $C^{\mathfrak{S}}_m$ be positive. For any $\delta > 0$, the protocol Π_s, constructed as above, is a $(C^{\mathfrak{S}}_{usk}, \delta)$ strongly-secure SKE protocol for the setup $\mathfrak{S}$, implying $C^{\mathfrak{S}}_{ssk} = C^{\mathfrak{S}}_{usk}$.*

Proof. See Appendix A.

Remark. Another approach to privacy amplification (step (iii)) without requiring randomness is to applying a deterministic (one-source) extractor. Assuming that the output distribution of the uniformly-secure SKE protocol is known, one can use deterministic extractors whose existence has been proved (cf. [10, Corollary 17.5]). The use of two-source extractors in the above is preferred as they are constructive and do not assume any knowledge about the distribution of S_{uA}, S_{uB}, and V_{uE} except those implied by the protocol definition.

5 Proof of Theorem 2: Weak to Strong Secrecy Capacity

We show this equality for the forward (Alice-to-Bob) direction of SMT; the backward direction can be shown similarly. Let $C^{\mathfrak{S}}_{wfs}$ be the weak forward secrecy capacity of the setup $\mathfrak{S}$. We give a four-step construction that for any $\delta > 0$, gives a $(C^{\mathfrak{S}}_{wfs}, \delta)$-strongly-secure forward SMT protocol Π_s by using a $(C^{\mathfrak{S}}_{wfs}, \delta')$-weakly-secure forward SMT protocol Π_w for sufficiently small δ'.

Similarly to Section 4.2, let K and N be sufficiently large integers and $L \leq \delta_1 NK$ and $r \geq 2NK(1-\delta_4)$ be integers as chosen in Appendix A. Let Π_w be capable of sending K bits of information from Alice to Bob, $M \in \{0,1\}^r$ be the message to be sent by Π_s, and the function $Inv : \{0,1\}^\rho \times \{0,1\}^r \to \{0,1\}^n \times \{0,1\}^n$, where $n = NK$ and $\rho = 2n - r$, be the inverter for the two-source extractor $TExt : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^r$ of Lemma 4.

Step (i): Message Expanding. Alice expands the message M to $(S_{A1}, S_{A2}) = Inv(Rnd, M)$, where $Rnd \in \{0,1\}^\rho$ is uniformly random.
* This step requires resources for ρ uniform bits, where $\rho = 2NK - r \leq 2NK\delta_4$.

Step (ii): Split and Sent. Alice splits S_{A1} and S_{A2} into N independent K-bit pieces, $(S_{A1,i})_{i=1}^2$ and $(S_{A2,i})_{i=1}^N$, and sends each piece independently using the weakly-secure protocol Π_w. Bob obtains message estimates $S_{B1} = (S_{B1,i})_{i=1}^N$ and $S_{B2} = (S_{B2,i})_{i=1}^N$. Eve's view of this is $(V_{E1}, V_{E2}) = (V_{E1,i}, V_{E2,i})_{i=1}^N$.
* This step requires resources for $2N$ repetition of Π_w that costs $2NCost_{\Pi_w}^{\mathfrak{S}}$.

Step (iii): Information Reconciliation. Alice finds error-correction functions h_1 and h_2 from a universal family of hash functions $\mathcal{H} : \{0,1\}^n \to \{0,1\}^L$. She calculates $h(S_{A1})$ and $h(S_{A2})$, and sends them reliably to Bob. Bob uses these to decode (S_{B1}, S_{B2}) into $(\hat{S}_{A1}, \hat{S}_{A2})$. Eve's view of this is (V_{rE1}, V_{rE2}).
* This step requires reliable transmission of $2L$ bits, which costs at most $2LCost_{\Pi_w}^{\mathfrak{S}}$.

Step (iv): Message Extraction. Bob calculates $\hat{M} = TExt(\hat{S}_{A1}, \hat{S}_{A2})$.

Proving this construction is strongly-secure requires the setup to allow local randomness for Alice to use in step (ii), message expansion. The proof is very similar to that of Appendix A for SKE; hence, omitted from the paper.

6 Concluding Remarks

Maurer and Wolf [15] proved the equality of weak and strong SK capacities for the setups in [1,9,14,21]. Their approach can also be applied to a general discrete memoryless setup as long as the following two conditions hold: (1) randomness is accessible, and (2) reliable transmission is possible. We have modified this approach and proved the above equality only requiring condition (2). We have also provided a proof for the equality of the weak and strong secrecy capacities in SMT, where we require condition (1). This shows a duality between SMT and SKE (see Table 1). In both cases, we have not shown whether these derived conditions are necessary. In the following, we argue against their necessity.

We argue that the condition of Theorem 1 is not always necessary. A trivial counterexample is a multiple-source $(\mathcal{X}_A, \mathcal{X}_B, \mathcal{X}_E)$ with per-use cost of 1 and distribution P_{X_A,X_B,X_E} that generates $X_A = X_B$ for Alice and Bob as well as X_E for Eve such that $H(X_A|X_E) = c$ for some constant $c > 0$. The setup has zero reliability capacity as there is no communication channel. However, Alice and Bob can establish a strongly-secure key of rate c by generating many source

Table 1. The requirements for proving the equality of capacities

Requirement	**MW approach** (SK capacity)	**Our approach** (SK capacity)	**Our approach** (Secrecy capacity)
Randomness access	required	-	required
Reliable transmission	required	required	-

outputs and applying a two-extractor. It is easy to show that both weak and strong SK capacities equal c. Similarly, the condition of Theorem 2 is not always necessary. Consider a scenario where Alice is connected to Bob through a one-way noisy channel that does not leak to Eve. Alice can send a message to Bob using deterministic coding and without requiring randomness. Both weak and strong secrecy capacities in this case equal the channel (reliability) capacity.

It is interesting to know whether there are non-trivial setups for which these conditions are not required, and to derive necessary and sufficient conditions for the equality of weak and strong secrecy/SK capacities.

References

1. Ahlswede, R., Csiszár, I.: Common randomness in information theory and cryptography. Part I: secret sharing. IEEE Transaction on Information Theory 39, 1121–1132 (1993)
2. Ahmadi, H., Safavi-Naini, R.: Secret key establishment over a pair of independent broadcast channels. In: International Symposium on Information Theory and its Application, pp. 185–190 (2010); Full version on the arXiv preprint server, arXiv:1001.3908
3. Ahmadi, H., Safavi-Naini, R.: New results on key establishment over a pair of independent broadcast channels. In: International Symposium on Information Theory and its Application, pp. 191–196 (2010); Full version on the arXiv preprint server, arXiv:1004.4334v1
4. Ahmadi, H., Safavi-Naini, R.: Common Randomness and Secret Key Capacities of Two-Way Channels. In: Fehr, S. (ed.) ICITS 2011. LNCS, vol. 6673, pp. 76–93. Springer, Heidelberg (2011)
5. Ahmadi, H., Safavi-Naini, R.: Secret Keys from Channel Noise. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 266–283. Springer, Heidelberg (2011)
6. Carter, L., Wegman, M.N.: Universal Classes of Hash Functions. Journal of Computer and System Sciences 18, 143–154 (1979)
7. Chor, B., Goldreich, O.: Unbiased bits from sources of weak randomness and probabilistic communication complexity. SIAM Journal on Computing 17, 230–261 (1988)
8. Cover, T.M., Thomas, J.: Elements of information theory, 2nd edn. Wiley-IEEE (2006)
9. Csiszár, I., Körner, J.: Broadcast channels with confidential messages. IEEE Transaction on Information Theory 24, 339–348 (1978)
10. Csiszár, I., Körner, J.: Information theory: coding theorems for discrete memoryless systems, 2nd edn., Cambridge (2011)

11. Csiszár, I., Narayan, P.: Common randomness and secret key generation with a helper. IEEE Transaction on Information Theory 46, 344–366 (2000)
12. Dodis, Y., Elbaz, A., Oliveira, R., Raz, R.: Improved Randomness Extraction from Two Independent Sources. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) APPROX and RANDOM 2004. LNCS, vol. 3122, pp. 334–344. Springer, Heidelberg (2004)
13. Khisti, A., Diggavi, S., Wornell, G.: Secret key generation with correlated sources and noisy channels. In: IEEE International Symposium on Information Theory, pp. 1005–1009 (2008)
14. Maurer, U.: Secret key agreement by public discussion from common information. IEEE Transaction on Information Theory 39, 733–742 (1993)
15. Maurer, U., Wolf, S.: Information-Theoretic Key Agreement: From Weak to Strong Secrecy for Free. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 351–368. Springer, Heidelberg (2000)
16. Mitrpant, C., Han Vinck, A.J., Luo, Y.: An achievable region for the Gaussian wiretap channel with side information. IEEE Transaction on Information Theory 52, 2181–2190 (2006)
17. Nisan, N., Zuckerman, D.: Randomness is linear in space. Journal of Computer and System Science 52, 43–52 (1996)
18. Prabhakaran, V., Eswaran, K., Ramchandran, K.: Secrecy via sources and channels - a secret key - secret message rate trade-off region. In: IEEE International Symposium on Information Theory, pp. 1010–1014 (2008)
19. Shannon, C.E.: Two-way communication channels. In: 4th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 611–644 (1961)
20. Tekin, E., Yener, A.: The general Gaussian multiple access channel and two-way wire-tap channels: achievable rates and cooperative jamming. IEEE Transactions on Information Theory 54, 2735–2751 (2008)
21. Wyner, A.D.: The wire-tap channel. Bell System Technical Journal 54, 1355–1367 (1975)

A Proof of Lemma 6: Analyzing Π_s

Prerequisites. We shall show that the protocol Π_s is a $(C^{\mathfrak{S}}_{usk}, \delta)$-strongly-secure SKE protocol, i.e., it satisfies reliability, strong secrecy, and strong uniformity as in Definition 6. The following analysis is given assuming that "Alice" sends the $2L$ bits of information for reconciliation. Similar analysis works when Bob sends information reconciliation bits. Using the reliability property of Π_u, we write

$$\forall 1 \leq i \leq N, 1 \leq j \leq 2: \quad \Pr(S_{uAj,i} \neq S_{uAj,i}) \leq \delta'.$$

Fano's inequality (cf. [8, Ch 2]) and the independence of Π_u repetitions give us

$$\forall 1 \leq j \leq 2: \quad H(S^N_{uAj}|S^N_{uBj}) \leq N\left(\delta' K + 1\right).$$

We combine the above with Lemma 3 to reach the following. For any $\delta_1 > 0$, by choosing ϵ_1 sufficiently small, there exists L satisfying

$$L \leq (1+\epsilon_1)N\left(\delta' K + 1\right) \leq \delta_1 NK, \tag{8}$$

for which information reconciliation succeeds with probability $\geq 1 - 2\epsilon_1$, i.e.,

$$\Pr(S'^N_{uA1} = S'^N_{uB1}) \geq 1 - \epsilon_1 \qquad \text{and} \qquad \Pr(S'^N_{uA2} = S'^N_{uB2}) \geq 1 - \epsilon_1. \tag{9}$$

The secrecy and uniformity properties of Π_u (see (5b) and (6)) imply

$$\forall 1 \leq i \leq N, 1 \leq j \leq 2: \quad H(S_{uAj,i}|V_{uEj,i}) \geq (1-\delta')\left(K - \delta' Cost^{\mathfrak{S}}_{\Pi_u}\right). \tag{10}$$

Strong Secrecy: Proving (7b). Eve's view $View_E$ of the protocol originates from the first two steps of the above construction, where resources in the setup are used. In step (i), she observes (V^N_{uE1}, V^N_{uE2}). We use Lemma 2, with ϵ_2 in place of ϵ, for the following. Let $\epsilon_2 \leq 1/(NK^2)$ so that the event $\mathcal{E}_j$ holds with probability $\Pr(\mathcal{E}_j) \geq 1 - \epsilon_2 \geq 1 - \frac{1}{NK^2}$. From the lemma, for all views v_j,

$$\begin{aligned} H_\infty(\tilde{S}_{jA}|V^N_{uEj} = v_j, \mathcal{E}_j) &\overset{(a)}{=} H_\infty(S^N_{uAj}|V^N_{uEj} = v_j, \mathcal{E}_j) \\ &\overset{(b)}{\geq} N\left((1-\delta')(K - \delta' Cost^{\mathfrak{S}}_{\Pi_u}) - \epsilon_2\right) \\ &\geq NK(1-\delta_2), \end{aligned} \tag{11}$$

for some $\delta_2 > 0$ that can be made arbitrarily small based on δ' and ϵ_2 when N and K are sufficiently large. Equality (a) is due to the injective binary mapping function and inequality (b) follows from (10) and Lemma 2.

In step (ii), Eve observes the independent variables V_{rEj}, for $1 \leq j \leq 2$, each of which reveals some information about the information reconciliation (error-correction) bits $h_j(S^N_{uAj})$. This implies the Markov chain $V_{rEj} \leftrightarrow h_j(S_{uAj,i}) \leftrightarrow (S^N_{uAj}, V^N_{uEj})$, which lets us write the following. For any instance of Eve's view v'_j there exists an L-bit string w such that, for $\epsilon_3 > 0$ and $1 \leq j \leq 2$, we have

$$\begin{aligned} H_\infty(\tilde{S}_{jA}|V^N_{uEj} = v_j, V_{rEj} = v'_j, \mathcal{E}_j) &\geq H_\infty(\tilde{S}_{jA}|V^N_{uEj} = v_j, h_j(S^N_{uAj}) = w, \mathcal{E}_j) \\ &\overset{(a)}{\geq} H_\infty(\tilde{S}_{jA}|V^N_{uEj} = v_j, \mathcal{E}_j) - L - \epsilon_3 \qquad \text{(with prob. } \geq 1 - 2^{-\epsilon_3}) \\ &\overset{(b)}{\geq} NK(1-\delta_1-\delta_2) - \epsilon_3 \qquad \text{(with prob. } \geq 1 - 2^{-\epsilon_3}). \end{aligned}$$

Inequality (a) follows from a trivial property of the min-entropy function (cf. [15]), and inequality (b) is due to (8), (11), and Lemma 3. We continue, by choosing $\epsilon_3 = \log(NK^2)$, which leads to

$$H_\infty(\tilde{S}_{jA}|V^N_{uEj} = v_j, V_{rEj} = v'_j, \mathcal{E}_j) \geq NK(1-\delta_3) \qquad \text{(with prob. } \geq 1 - \tfrac{1}{NK^2}),$$

for arbitrarily small $\delta_3 > 0$, by choosing N and K sufficiently large. The above inequality shows that, with high probability for any instance of Eve's view, each $\tilde{S}_{jA}$ is arbitrarily close to uniform in rate. Applying the two-source extractor makes the output arbitrarily close to uniform in terms of its absolute value. This is shown in the following. For sufficiently small $\epsilon_4 > 0$, let $\gamma = 2^{-\epsilon_4 NK}$ and the extractor output length r satisfy

$$r = 3NK(1-\delta_3) - NK - 4\log(1/\epsilon) \geq 2NK(1-\delta_4),$$

for $\delta_4 > 0$ which can be made arbitrarily small by choosing N and K sufficiently large. According to Lemma 4, the output $\tilde{S}_A$ of the two-source extractor given $View_E = v_e$, $\mathcal{E}_1$, and $\mathcal{E}_2$ is γ-close to uniform with probability $1 - \frac{1}{NK}$. Similarly to [15, Lemma 6], the above implies

$$H(\tilde{S}_A|View_E = v_e, \mathcal{E}_1, \mathcal{E}_2) \geq r - 2^{\epsilon_4 NK} \qquad \text{(with prob. } \geq 1 - \tfrac{1}{NK^2}\text{)}.$$

By taking into consideration the probabilities of $\mathcal{E}_1$ and $\mathcal{E}_2$, we arrive at

$$\begin{aligned} H(\tilde{S}_A|View_E) &\geq (1 - \frac{1}{NK^2} + \Pr(\overline{\mathcal{E}}_1) + \Pr(\overline{\mathcal{E}}_2))(r - 2^{\epsilon_4 NK}) \\ &\geq (1 - \frac{3}{NK^2})(r - 2^{\epsilon_4 NK}) \geq r - \delta_5, \end{aligned} \tag{12}$$

for $\delta_5 < \delta$, by choosing N and K sufficiently large. This proves strong secrecy as $r = \log|\mathcal{S}|$.

Reliability: Proving (7a). We first note that (12) also shows $H(\tilde{S}_A) \geq r - \delta_5$, i.e., the absolute entropy of $\tilde{S}_A$ can be made arbitrarily close to uniform. Similarly to [15], the uniformization step converts $\tilde{S}_A$ to a completely uniform variable S_A with error probability $\Pr(S_A \neq \tilde{S}_A) \leq \epsilon_5$ for arbitrarily small $\epsilon_5 > 0$. The following steps eventually prove the reliability property by using (9).

$$\begin{aligned} \Pr(S_A \neq S_B) &\leq \Pr(S_A \neq \tilde{S}_A \ \vee \ \tilde{S}_A \neq S_B) \leq \Pr(S_A \neq \tilde{S}_A) + \Pr(\tilde{S}_A \neq S_B) \\ &\leq \Pr(S_A \neq \tilde{S}_A) + \Pr(S'^N_{uA1} \neq S'^N_{uB1}) + \Pr(S'^N_{uA2} \neq S'^N_{uB2}) \leq \epsilon_5 + 2\epsilon_1 < \delta, \end{aligned}$$

for appropriately small ϵ_1 and ϵ_5 when N and K are sufficiently large.

Strong Uniformity: Proving (7c). The protocol Π_s provides r-bit secret key. Obviously perfect uniformity holds as S_A is uniformly distributed thanks to the uniformization step. The rest is to analyze the key rate and show that it is arbitrarily close to the uniform SK capacity $C^{\mathfrak{S}}_{usk}$. Before all, note that the protocol Π_u satisfies the randomness condition (5c), i.e., $K/Cost^{\mathfrak{S}}_{\Pi_u} \geq C^{\mathfrak{S}}_{usk} - \delta'$.

The cost $Cost^{\mathfrak{S}}_{\Pi_s}$ of the protocol Π_s equals those of steps (i) and (ii), i.e., $2NCost^{\mathfrak{S}}_{\Pi_u}$ plus the cost of sending $2L$ information bits by either party. We note that other steps do not use any communication resource and hence have no effect on the total cost. We recall from Definition 2 that, for any $\epsilon_6 > 0$ and large enough N and K (hence L), there exists a protocol that send $2L$ information bits in either (forward or backward) direction with error probability ϵ_6 and cost at most $\frac{2L}{C^{\mathfrak{S}}_m - \epsilon_6}$. The SK rate achieved by the protocol Π_s is obtained as follows.

$$\begin{aligned} \frac{H(S_A)}{Cost^{\mathfrak{S}}_{\Pi_s}} = \frac{r}{Cost^{\mathfrak{S}}_{\Pi_s}} &\geq \frac{2NK(1-\delta_4)}{2NCost^{\mathfrak{S}}_{\Pi_u} + 2L/(C^{\mathfrak{S}}_m - \epsilon_6)} \geq \frac{2NK(1-\delta_4)}{2NCost^{\mathfrak{S}}_{\Pi_u} + 2\delta_1 NK/(C^{\mathfrak{S}}_m - \epsilon_6)} \\ &\geq \frac{Cost^{\mathfrak{S}}_{\Pi_u}(C^{\mathfrak{S}}_{usk} - \delta')(1-\delta_4)}{Cost^{\mathfrak{S}}_{\Pi_u} + \delta_1 Cost^{\mathfrak{S}}_{\Pi_u} C^{\mathfrak{S}}_{usk}/(C^{\mathfrak{S}}_m - \epsilon_6)} = \frac{(C^{\mathfrak{S}}_{usk} - \delta')(1-\delta_4)}{1 + \delta_1 C^{\mathfrak{S}}_{usk}/(C^{\mathfrak{S}}_m - \epsilon_6)} \geq C^{\mathfrak{S}}_{usk}(1-\delta), \end{aligned}$$

The last inequality is attained by choosing $\epsilon_6, \delta_1, \delta_4$, and δ' suitably small for sufficiently large N and K. This proves that the strong SK capacity equals to the uniform SK capacity.

COMPASS: Authenticated Group Key Agreement from Signcryption

Nick Mailloux[1], Ali Miri[1,2], and Monica Nevins[1]

[1] Department of Mathematics and Statistics
University of Ottawa, Ottawa, ON, Canada
nickmailloux@gmail.com, mnevins@uottawa.ca
[2] Department of Computer Science
Ryerson University, Toronto, ON, Canada
Ali.Miri@ryerson.ca

Abstract. In this paper, we propose a new authenticated group key agreement protocol that uses identity-based signcryption to achieve the optimal communication complexity of a single broadcast message per member in a single round of communication, set by Becker and Wille. Our protocol is provably secure in the random oracle model, provided that the signcryption scheme is secure. By choosing a signcryption scheme that satisfies some additional criteria, our protocol provides key integrity in an efficient manner.

Keywords: Authenticated group key agreement, communication efficiency, identity-based cryptography, key integrity, signcryption.

1 Introduction

Communication efficiency is one of the fundamental goals in the design of group key agreement (GKA) protocols. Our main concern is to reduce the number of rounds of communication, while also reducing the number of broadcast messages that must be sent and received by group members in each round. [4] derived lower bounds of n broadcast messages in only a single round of communication for contributory group key agreement protocols, where n is the size of the multicast group.

[6] proposed an efficient group key agreement protocol in which the multicast group consists of a group of *responders*, that broadcast their key contributions in the clear, and a distinguished *group leader*, that sends her secret contribution to each responder in some confidential and authenticated manner. The group session key is computed using the public contributions of the responders and the secret contribution of the group leader. This type of protocol is often referred to as a *computationally asymmetric group key agreement* protocol, since each group member contributes equally to the session key, but most of the computational burden is shifted to the group leader.

Prior to the work of [27], confidentiality and authentication were solely provided by encryption and signature schemes, respectively. Zheng proposed a new cryptographic primitive known as *signcryption*, that provides both confidentiality and authentication at a lower cost than the sequential composition of

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 95–114, 2013.

encryption and signature schemes. However, [1] suggested that efficiency may not be our only concern when designing such a scheme and proposed that the term signcryption refer to *any* secure scheme that provides both confidentiality and authentication in the public key setting.

Around the same time, [5] proposed the first practical and provably-secure *identity-based encryption* (IBE) scheme. Identity-based cryptography, proposed by [21], simplifies the issue of certificate management in the public key setting by allowing Alice's public key to be any string that uniquely identifies her, such as an email address. This approach requires a trusted third party, known as the private key generator (PKG), to generate the public system parameters and distribute the private keys corresponding to each user's identity using some master secret key. As a result, there is no need for a certificate authority (CA) to issue digital certificates binding Alice's public key to her identity. This quality is particularly convenient for the purposes of signcryption, since Bob can decrypt and verify the authenticity of Alice's ciphertext without communicating with a third party. Consequently, a better part of the signcryption schemes proposed in the literature are based on some identity-based public key infrastructure (ID-PKI), such as the schemes from [3,8,13,24].

1.1 Our Contribution

In this paper, we propose a computationally asymmetric *authenticated group key agreement* (AGKA) protocol that achieves the optimal level of communication efficiency set in [4]. Our protocol requires the responders to publicly broadcast their signed key contributions, while the group leader uses signcryption to transport her secret key contribution in a confidential and authenticated manner. Consequently, we refer to our protocol as the COMPutationally Asymmetric Signcryption Scheme based AGKA protocol, or the COMPASS protocol for short. We reduce the security of the protocol to the confidentiality and unforgeability of the signcryption scheme in the random oracle model.

While *any* secure signcryption scheme may be used in the COMPASS protocol, we choose an identity-based signcryption scheme for two reasons in particular. Firstly, since signcryption schemes thrive in an identity-based environment, this will produce the most efficient protocol. Secondly, since many of the signcryption schemes from the literature are based on an ID-PKI, we have a wealth of schemes to choose from. Furthermore, to produce the most efficient and secure protocol as possible, we choose an identity-based signcryption scheme that satisfies some specific properties in terms of scalability and signature verifiability. These properties also allow us to provide many of the strong contributory properties proposed in [2], such as verifiable contributiveness and key integrity, which will be discussed further in Section 2.3.

1.2 Related Work

[7] proposed the first provably-secure group key agreement protocol that meets the lower bound of [4]. Similar to the COMPASS protocol, their protocol takes a

computationally asymmetric approach to group key agreement. However, their protocol is less efficient, in part because they require the group leader to use separate encryption and signature schemes to securely transport her secret key contribution to the responders. In addition, their protocol was shown to be vulnerable to unknown key share attacks by [16].

Recently, [26] proposed a provably-secure one round asymmetric AGKA, in which messages are encrypted using a publicly computable encryption key. They use the term asymmetric in a different context, to refer to the fact that each user has their own secret decryption key. Other recent AGKA protocols can be found in [11] and [18].

The notion of using signcryption schemes in group key establishment has been considered before. Shortly after introducing signcryption to the cryptographic community, [28] proposed the idea of using signcryption schemes for authenticated key establishment. They presented several different protocols for signcryption-based key distribution and key agreement, including one for the case of multicast group communication. However, their protocol requires a group leader to choose a session key and signcrypt it for each responder, taking a distributive approach to key establishment. To the best of our knowledge, the COMPASS protocol is the first efficient and secure AGKA protocol that can be implemented using any secure signcryption scheme.

1.3 Organization of the Paper

The structure of this paper is as follows: In Section 2, we provide some preliminary definitions, including definitions of identity-based signature and signcryption schemes and the formal model used to prove the security of the COMPASS protocol. In Section 3, we define several desirable properties of signcryption schemes and their effect on our protocol. We present the proposed COMPASS protocol and compare the communication efficiency to other GKA protocols in Section 4. In Section 5, we provide a formal proof of the security of our protocol. In Section 6, we provide a concrete example to illustrate the COMPASS protocol using an identity-based signcryption scheme from the literature, and discuss some open problems and future work in Section 7.

2 Preliminaries

2.1 Identity-Based Signcryption

An identity-based public key infrastructure (ID-PKI), such as that of [5], simplifies the management of public key certificates and is the ideal setting for the COMPASS protocol. Furthermore, it is both efficient and convenient to use identity-based signature and signcryption schemes that employ the same ID-PKI in the COMPASS protocol. We formally define the concepts of identity-based signature and signcryption schemes here.

Definition 1. *An* identity-based signature scheme $\mathcal{S}$ *is characterized by three randomized algorithms* Setup, Extract *and* Sign, *and a deterministic algorithm* Verify. *They are described as follows:*

- $\mathsf{Setup}(\ell)$: Given a security parameter ℓ, a trusted authority known as the private key generator (PKG) generates the pair $\langle \mathsf{msk}, \mathsf{params} \rangle$ at random, where msk is the master secret key, known only to the PKG, and params refer to the public system parameters. These parameters include a global public key P_{pub}.
- $\mathsf{Extract}(\mathsf{params}, \mathsf{msk}, \mathsf{ID})$: Upon input of params, the master secret key msk and an identity string $\mathsf{ID} \in \{0,1\}^*$, the PKG returns the private signing key d_{ID} corresponding to the user ID.
- $\mathsf{Sign}(\mathsf{params}, d_{\mathsf{ID}}, m)$: Upon input of params, the signer's private signing key d_{ID} and a plaintext message $m \in \mathcal{M}$, the algorithm returns the signature σ.
- $\mathsf{Verify}(\mathsf{params}, \mathsf{ID}, m, \sigma)$: Upon input of params, the public key ID and a signature σ, the algorithm returns $\top$ if σ is a valid signature by the sender ID on the message m and $\perp$, otherwise.

The above algorithms must satisfy the standard consistency constraint that, given a signature $\sigma = \mathsf{Sign}(\mathsf{params}, d_{\mathsf{ID}}, m)$, we have $\top = \mathsf{Verify}(\mathsf{params}, \mathsf{ID}, m, \sigma)$.

Definition 2. *An* identity-based signcryption scheme $\mathcal{SC}$ *consists of three randomized algorithms* Setup, Extract *and* Sign/Encrypt, *and one deterministic algorithm* Decrypt/Verify. *The algorithms* Setup *and* Extract *comprising the ID-PKI are as defined in Definition 1. The remaining algorithms are described as follows:*

- $\mathsf{Sign/Encrypt}(\mathsf{params}, d_{\mathsf{A}}, \mathsf{ID}_b, m)$: Upon input of params, the private key d_A of the sender A, the identity ID_B of the recipient B and a plaintext message $m \in \mathcal{M}$, the algorithm returns the signcrypted ciphertext c.
- $\mathsf{Decrypt/Verify}(\mathsf{params}, d_B, \mathsf{ID}_A, c)$: Upon input of params, the private key d_B of the recipient B, the identity ID_A of the sender A and the signcrypted ciphertext c, the algorithm returns the message m if c decrypts into a message bearing A's signature, or $\perp$ otherwise.

The above algorithms must satisfy the standard consistency requirement that, given a ciphertext $c = \mathsf{Sign/Encrypt}(\mathsf{params}, d_A, \mathsf{ID}_B, m)$, we have that $m = \mathsf{Decrypt/Verify}(\mathsf{params}, d_B, \mathsf{ID}_A, c)$.

2.2 Group Communication Model

We use the model of [9], which is the conventional model for authenticated group key establishment and was used to prove the security of the GKA protocol in [7]. The following definitions are adapted from [9] to reflect the use of an ID-PKI.

Participation. We assume there is a fixed set of potential participants $\mathcal{U} = \{U_0, U_1, ..., U_\eta\}$, where the number of users is polynomial in the security parameter ℓ. Each participant $U_i \in \mathcal{U}$ is associated with a unique identity $\mathsf{ID}_i \in \{0,1\}^*$. The model allows different *instances* of a user, known as oracles, to be involved in distinct, but possibly concurrent, executions of the protocol. We denote instance α of a particular user U_i by Π_i^α and of a general user U by Π.

Initialization. Before the protocol is initialized, the PKG generates the global parameters params and master secret key msk using the Setup algorithm. Afterwards, the PKG issues a public/private key pair to each user $U_i \in \mathcal{U}$ using the $\mathsf{Extract}$ algorithm. We assume that the public key set is known by all participants.

Adversarial Model. We allow the adversary $\mathcal{A}$ to control all communication between instances of users. While interacting with the various instances, we allow the adversary to have certain capabilities that correspond to real world abilities or attacks. We model these capabilities by allowing the adversary to issue the following queries:

- $\mathsf{Extract}(U_i)$: Returns the private key d_i of user U_i obtained from the $\mathsf{Extract}$ algorithm.
- $\mathsf{Send}(\Pi_U^\alpha, M)$: Sends the message M to instance α of user U and returns the response generated by the oracle.
- $\mathsf{Execute}(U_{i_1}, ..., U_{i_n})$: Executes the protocol for a group of users $\{U_{i_1}, ..., U_{i_n}\}$, chosen by the adversary, and returns the transcript of the execution.
- $\mathsf{Reveal}(\Pi_U^\alpha)$: Returns the session key K if oracle Π_U^α has accepted the session key.
- $\mathsf{Corrupt}(U)$: Returns the private key d_U of user U, but does not reveal any internal data of any instance of U.
- $\mathsf{Test}(\Pi_U^\alpha)$: Models the semantic security of a session key and may only be asked once. It generates a random bit $b \in \{0,1\}$. Oracle Π_U^α returns the session key K if $b = 1$ and a random session key if $b = 0$.

The security model allows for both passive and active adversaries. A *passive adversary* is given access to the $\mathsf{Extract}$, $\mathsf{Execute}$, Reveal, $\mathsf{Corrupt}$ and Test queries. An *active adversary* is additionally given access to the Send query.

2.3 Security Definitions

Partnering. In our protocol, every message is broadcast to all users in the session. As a result, we define the notion of partnering as it is given in [9], using *session IDs* and *partner IDs*. A session ID for an oracle Π_U^α is denoted sid_U^α and is equal to the concatenation of all messages that are sent and received by instance Π_U^α in the execution of the protocol. A partner ID for an oracle Π_U^α is denoted pid_U^α and consists of the identities of all the users establishing a key in the α^{th} session. We say that oracles Π_i^α and Π_j^β are *partnered* if, and only if, $\mathsf{sid}_i^\alpha = \mathsf{sid}_j^\beta$ and $\mathsf{pid}_i^\alpha = \mathsf{pid}_j^\beta$.

Freshness. An oracle Π_i^α is said to be *fresh* (or hold a *fresh* session key K) if it satisfies the following:

1. Π_i^α has accepted the session key K and the adversary did not issue $\mathsf{Reveal}(\Pi_i^\alpha)$, or $\mathsf{Reveal}(\Pi_j^\beta)$ for any oracle Π_j^β partnered with Π_i^α
2. the adversary has not issued a $\mathsf{Corrupt}$ query for user U_i before a query of the form $\mathsf{Send}(\Pi_i^\alpha, *)$, or $\mathsf{Send}(\Pi_j^\beta, *)$ for any oracle Π_j^β partnered with Π_i^α.

Security Game. The security of a protocol $\mathcal{P}$ is defined in terms of the following game between the adversary $\mathcal{A}$ and an infinite set of oracles Π_U^α, for $U \in \mathcal{U}$ and $\alpha \in \mathbb{N}$:

- Phase 1: The adversary $\mathcal{A}$ issues queries defined above to the oracles. Afterwards, $\mathcal{A}$ issues the Test query to a fresh oracle Π_i^α.
- Challenge: The challenger $\mathcal{C}$ responds with the session key K_b as per the Test query.
- Phase 2: $\mathcal{A}$ continues to query the oracles, but may not issue a $\mathsf{Corrupt}$ query for any user, or the Reveal query for oracle Π_i^α or any oracle Π_j^β partnered with Π_i^α.
- Response: The adversary outputs a guess b' as to the value of the bit b. The adversary wins the game if $b' = b$.

We measure the adversary's advantage in the security game by her ability to distinguish between the session key and a random value. The advantage of $\mathcal{A}$ in attacking $\mathcal{P}$ is defined as

$$\mathsf{Adv}_{\mathcal{A},\mathcal{P}}(\ell) = |2 \cdot \Pr[b' = b] - 1|$$

for the security parameter ℓ.

Secure AGKA Protocol. We say that a protocol $\mathcal{P}$ is a *secure authenticated group key agreement protocol* if it satisfies the following properties:

1. *Validity*: Partner oracles Π_i^α and Π_j^β accept the same session key in the presence of a passive adversary $\mathcal{A}$.
2. *Indistinguishability*: The advantage $\mathsf{Adv}_{\mathcal{A},\mathcal{P}}(\ell)$ of any probabilistic polynomial time (PPT) active adversary $\mathcal{A}$ is negligible.

Contributory Protocols. Contributivity properties of group key agreement protocols have been discussed in [2]. A group key agreement protocol $\mathcal{P}$ is said to be *contributory* if each user contributes to the session key equally, ensuring that no user can predetermine the session key. A contributory protocol $\mathcal{P}$ provides *group integrity* if each party is assured of every other party's participation. A contributory protocol $\mathcal{P}$ provides *verifiable contributiveness* if each session participant is assured of all other participants' contributions to the session key; in particular this implies group integrity, since assurance of a participant's contribution implies assurance of their participation. Finally, a contributory protocol $\mathcal{P}$ is said to provide *key integrity* if each participant is assured that the session key is a function of *only* the contributions of *all* valid participants. Clearly, key integrity implies verifiable contributiveness.

3 Selecting a Signcryption Scheme

To produce the most efficient and secure protocol possible, we choose a signcryption scheme that satisfies some specific properties. We define the various signcryption properties here and discuss the effect each property has on the protocol.

3.1 Multi-receiver Signcryption

A signcryption scheme provides *efficient support for multiple recipients* and is known as a *multi-receiver signcryption scheme*, if the sender A can signcrypt a message for a group of recipients without executing the Sign/Encrypt algorithm for each individual recipient. A number of multi-receiver identity-based signcryption schemes have been proposed in the literature. [8] showed that his identity-based signcryption scheme can be extended to support multiple recipients, while [24] proposed an efficient multi-receiver variant of the identity-based signcryption scheme of [13] .

Using a multi-receiver signcryption scheme, we reduce the computational costs of the group leader in the COMPASS protocol.

3.2 $\mathcal{S}$-Verifiable Signcryption

A signature scheme provides *non-repudiation* if the recipient of a message can prove to a third party that the message was indeed signed by the sender. It is more difficult to provide this property in signcryption schemes, without requiring the recipient to divulge their long-term private key. A signcryption scheme is said to be $\mathcal{S}$-*verifiable* [23] for some signature scheme $\mathcal{S}$, if the Decrypt/Verify algorithm outputs a *detachable signature* σ for the signcrypted message that the recipient can verify using $\mathcal{S}$. For example, the identity-based signcryption schemes of [13,24] produce a detachable signature for the identity-based signature scheme of [12]. Since non-repudiation results from the unforgeability of $\mathcal{S}$, the detachable signature may be forwarded to a third party for verification, providing non-repudiation in the signcryption scheme.

The property of $\mathcal{S}$-verifiability is used to simplify the authentication of member contributions in the COMPASS protocol. If the group leader uses an $\mathcal{S}$-verifiable signcryption scheme and the responders use the signature scheme $\mathcal{S}$, the detachable signature for the leader's secret contribution and the signatures for the responders' public contributions can all be authenticated using the Verify algorithm of $\mathcal{S}$.

3.3 Batch Verification

Informally speaking, a signature scheme is *batch verifiable* if a recipient can verify a set of signatures without executing the Verify algorithm for each individual signature. As discussed in [17], a batch verification algorithm should detect with high probability whether each individual signature in the batch is valid.

A number of batch-verifiable identity-based signature schemes were proposed in [17], including a batch verifiable variation of the Cha-Cheon scheme.

Using a batch verifiable signature scheme $\mathcal{S}$ in the COMPASS protocol, each member efficiently verifies the signatures of every other member at once, substantially reducing the computational costs of the protocol. Furthermore, since each member is assured of the authenticity of every key contribution, we provide the property of key integrity.

3.4 Multipurpose Signatures

Most signature schemes from the literature require the input of some randomized ephemeral data, say r, and produce a signature σ containing an element X that is authentically linked to the signer and is produced as a function of the value r. For example, in the Cha-Cheon signature scheme, a signer with public/private key pair $(Q_{\mathsf{ID}}, d_{\mathsf{ID}})$ produces a signature of the form $\sigma = (X, Z)$, where any third party can authenticate the message using $X = rQ_{\mathsf{ID}}$ and Z. We refer to such a signature as a *multipurpose signature* with ephemeral public value X.

By using a scheme $\mathcal{S}$ that produces multipurpose signatures in the COMPASS protocol, the responders can use the ephemeral public value X as their publicly broadcast key contribution, rather than choosing a random string N_i and producing a signature for N_i. As a result, the responder need only broadcast a single multipurpose signature, rather than a signature and a contribution, reducing the size of the broadcast message.

4 COMPASS Authenticated Group Key Agreement

4.1 The COMPASS Protocol

Let $\mathcal{SC}$ be an identity-based signcryption scheme, as given in Definition 2, providing the properties discussed in Section 3. In particular, let $\mathcal{SC}$ be $\mathcal{S}$-verifiable for some identity-based signature scheme $\mathcal{S}$, as given in Definition 1, with batch verification algorithm Batch.

Suppose we have group of $n + 1$ members $\mathcal{U} = \{U_0, ..., U_n\}$, where member U_0 is the group leader and the rest of the members belong to the group of responders $\mathcal{R}$. Using the Setup and Extract algorithms of the ID-PKI, each group member $U_i \in \mathcal{U}$ is associated with the public/private key pair (ID_i, d_i). To begin group communication, the group leader sends out some session identifier, say $session_{ID} \in \{0,1\}^*$. A summary of the protocol is given in Table 1. The COMPASS protocol runs in a single round of communication, as follows:

Round 1. Each responder $U_i \in \mathcal{R}$ signs the message "$\mathcal{U} \parallel session_{ID}$"[1] using the Sign algorithm of $\mathcal{S}$ to obtain the multipurpose signature σ_i containing ephemeral public value X_i. Member U_i broadcasts the signature σ_i to the rest of the group.

[1] By requiring each member to sign the session identifier $session_{ID}$ and the multicast group $\mathcal{U}$, we provide resilience to insider impersonation and unknown key share attacks, as shown in [16].

Table 1. Summary of the COMPASS Protocol

COMPASS
Round 1 $U_i : \sigma_i \leftarrow \mathsf{Sign}(\text{“}\mathcal{U} \parallel session_{ID}\text{”}, d_i),\ 1 \leq i \leq n$ $U_i \xrightarrow{B} \mathcal{U} : \sigma_i,\ 1 \leq i \leq n$ $U_0 : N \xleftarrow{r} \{0,1\}^*$ $U_0 : c_i \leftarrow \mathsf{Sign/Encrypt}(\text{“}\mathcal{U} \parallel session_{ID} \parallel N\text{”}, d_0, \mathsf{ID}_i), 1 \leq i \leq n$ $U_0 \xrightarrow{B} \mathcal{U} : \langle c_1, ..., c_n \rangle$
Key Computation $U_i : \langle \text{“}\mathcal{U} \parallel session_{ID} \parallel N\text{”}, \sigma_0 \rangle \leftarrow \mathsf{Decrypt/Verify}(c_i, d_i, \mathsf{ID}_0), 1 \leq i \leq n$ U_i : $\mathsf{Batch}((\text{“}\mathcal{U} \parallel session_{ID} \parallel N\text{”}, \sigma_0, \mathsf{ID}_0),$ $(\text{“}\mathcal{U} \parallel session_{ID}\text{”}, \sigma_1, \mathsf{ID}_1), ..., (\text{“}\mathcal{U} \parallel session_{ID}\text{”}, \sigma_n, \mathsf{ID}_n)),\ 0 \leq i \leq n$ $K = H_1(X_1, ..., X_n) \oplus H_2(\mathcal{U} \parallel session_{ID} \parallel N)$

The group leader U_0 chooses a secret contribution $N \in_R \{0,1\}^*$ and signcrypts the message “$\mathcal{U} \parallel session_{ID} \parallel N$” for each responder using the Sign/Encrypt algorithm of $\mathcal{SC}$, producing the ciphertext set $\langle c_1, ..., c_n \rangle$. She broadcasts the set of signcrypted ciphertexts, along with a list $\mathcal{L}$ indicating which ciphertext c_i corresponds to which responder U_i.

Key Computation. Upon receiving the set of signcrypted values from the group leader, each responder U_i extracts their specific ciphertext c_i and, using their long-term private key d_i, the long-term public key of the leader ID_0 and the Decrypt/Verify algorithm of $\mathcal{SC}$, decrypts it to recover the leader's secret contribution N and the detachable signature σ_0.

Each group member batch verifies the signatures $\{\sigma_0, ..., \sigma_n\}$ using the Batch algorithm. If the verification succeeds, they compute the session key as a function of the public key contributions $\{X_1, ..., X_n\}$ of the responders and the secret contribution N of the group leader. The session key is given by

$$K = H_1(X_1, ..., X_n) \oplus H_2(\mathcal{U} \parallel session_{ID} \parallel N),$$

for some cryptographic hash functions H_1 and H_2, which will be modelled as random oracles.

4.2 Efficiency of the COMPASS Protocol

We compare the efficiency of the COMPASS protocol with three of the more prominent GKA protocols from the literature: the Tree-based Group Diffie-Hellman (TGDH) protocol of [19], the Burmester-Desmedt (BD) protocol of [10]

Table 2. Communication costs and group key properties compared with the TGDH, BD and CKA protocols. We use $\Diamond$ to denote that the protocol does not require the authentication of all contributions.

		COMPASS	TGDH	BD	CKA
Rounds (Max)		1	$\lceil \log_2 (n+1) \rceil$	2	1
Messages (Total)	Unicast	–	–	–	–
	Multicast	$n+1$	$2n$	$2(n+1)$	$n+1$
Contributory Key		✓	✓	✓	✓
Authentication		✓	–	–	$\checkmark^{\Diamond}$
Group Integrity		✓	×	×	✓
Verifiable Contributiveness		✓	×	×	×
Key Integrity		✓	×	×	×

and the Conference Key Agreement (CKA) protocol of [7] in terms of communication costs and key agreement properties in Table 2. The exact computational costs of the protocol will ultimately depend on the choice of signcryption scheme. In Section 6, we provide a concrete example of the COMPASS protocol and provide a thorough account of the computational costs in this case.

The COMPASS protocol runs in a single round of communication and requires only one broadcast message by each group member, which meets the optimal communication complexity for group key agreement. The CKA protocol achieves the same communication efficiency, while the BD protocol requires twice as many messages sent in twice as many rounds. The TGDH protocol does not provide constant round complexity, requiring as many rounds as the height of the binary key tree, and a total of $2n$ broadcast messages.

Since the signcryption scheme $\mathcal{SC}$ and the signature scheme $\mathcal{S}$ provide unforgeability, the batch verification stage ensures that all contributions are authentic and have not been tampered with by an adversary, and that each member is in possession of the same session string. Since each member U_i computes K as a product of the authenticated responder contributions $\{X_1,...,X_n\}$, the authenticated leader contribution N and the session string, they are assured that the session key K has been authentically contributed to by every member in the group. Furthermore, since the session key is computed using *only* the authenticated values, they are assured that an adversary has not introduced any extraneous contributions. As a result, the COMPASS protocol provides the strongest contributory property of key integrity. Since the TGDH and BD protocols do not provide authentication, they cannot guarantee these properties. The CKA protocol provides group integrity, since the key is computed using only member contributions. However, since the protocol does not require authentication of responder contributions, it does not provide verifiable contributiveness or key integrity.

5 Security Proof

To prove that the COMPASS protocol is a secure authenticated group key agreement protocol, we must show that it satisfies the validity and indistinguishability

requirements from Section 2.3. Since the validity of the protocol is straightforward, we show that the advantage of any PPT active adversary attacking the COMPASS protocol is negligible.

Since signcryption combines the concepts of encryption and signature schemes, the standard notions of security for identity-based signcryption schemes are *existential unforgeability under adaptive chosen message attacks* (EUF-IBSC-CMA) and *indistinguishability under adaptive chosen ciphertext attacks* (IND-IBSC-CCA2), as discussed in [13]. We prove the security of the COMPASS protocol using the formal model for analyzing identity-based signcryption schemes from [13], by showing that an adversary has no advantage in breaking a simulation of the scheme. We allow the adversary to submit a set of recipients (where the identity ID of the sender is different from the set of receiver identities $\{\mathsf{ID}_{i_1}, ..., \mathsf{ID}_{i_n}\}$) in the Sign/Encrypt query to account for multi-receiver signcryption.

Let $\mathsf{Adv}_{\mathsf{COMPASS}}(t, q_{ex})$ denote the maximum advantage of any adversary attacking the COMPASS protocol in running time t and making q_{ex} Execute queries. We suppose that $\mathcal{SC}$ is an $\mathcal{S}$-verifiable identity-based signcryption scheme that is both EUF-IBSC-CMA secure and IND-IBSC-CCA2 secure in the multi-receiver signcryption mode. We suppose that the hash functions H_1 and H_2 are modelled as random oracles.

Theorem 1. *The proposed* COMPASS *protocol with identity-based signcryption scheme $\mathcal{SC}$ is a secure AGKA protocol. Specifically,*

$$\mathsf{Adv}_{\mathsf{COMPASS}}(t, q_{ex}) \leq (n+1)\, q_{ex} \mathsf{Adv}_{\mathcal{SC}}^{\mathsf{Distinguish}}(t) + (n+1)\, \mathsf{Adv}_{\mathcal{SC}}^{\mathsf{Forge}}(t),$$

where $\mathsf{Adv}_{\mathcal{SC}}^{\mathsf{Forge}}(t)$ is the maximum advantage of any EUF-IBSC-CMA forger $\mathcal{F}$ of the signcryption scheme $\mathcal{SC}$ and $\mathsf{Adv}_{\mathcal{SC}}^{\mathsf{Distinguish}}(t)$ is the maximum advantage of any IND-IBSC-CCA2 algorithm $\mathcal{D}$ attacking the indistinguishability of the signcryption scheme $\mathcal{SC}$, all running in time t.

Proof. Following the approach of [7], we show that an adversary $\mathcal{A}$ with a non-negligible advantage $\mathsf{Adv}_{\mathsf{COMPASS}}$ in breaking a simulation of the COMPASS protocol can be used to break the security of the underlying signcryption scheme. Informally speaking, we simultaneously play the role of the adversary in the EUF-IBSC-CMA or IND-IBSC-CCA2 security game, while also playing the role of the set of oracles in the AGKA security game with the COMPASS adversary. If the adversary has any advantage, we use it to gain an advantage in the signcryption security games.

We divide the proof into two separate cases. We assume that the adversary gains her advantage by either (1) forging the leader's signature within the ciphertext for the signcryption scheme $\mathcal{SC}$ or (2) breaking the protocol without altering authentication transcripts. In case (1), we use the adversary $\mathcal{A}$ to construct a forging algorithm $\mathcal{F}$ for the signcryption scheme $\mathcal{SC}$. In case (2), we use the adversary to build a distinguishing algorithm $\mathcal{D}$ to attack the semantic security of $\mathcal{SC}$.

To simplify the notation, we assume that the multicast group $\mathcal{U} = \{U_0, ..., U_n\}$ remains the same from one session to another. The dynamic case is handled

similarly. We allow the adversary to choose the leader from the group $\mathcal{U}$ for each session. To indicate the different possible roles of the members, we use the notation $\{U_{i_0}, U_{i_1}, ..., U_{i_n}\}$ to represent the multicast group of the i^{th} session, where each U_{i_j}, $0 \leq j \leq n$, corresponds to some $U_k \in \mathcal{U}$. To be as general as possible, we assume that we do not have prior knowledge of the adversary's choice for the group leader.

Signature within the Ciphertext Forgery on $\mathcal{SC}$. Suppose an adversary $\mathcal{A}$ gains her advantage by forging the signed contribution (within the ciphertext) of the group leader, thereby fooling the responders into sharing a session key with her. We can use the adversary to construct a forger $\mathcal{F}$ for the signcryption scheme $\mathcal{SC}$ with non-negligible advantage in the EUF-IBSC-CMA security game.

Lemma 1. *Let* Forge *be the event that an adversary $\mathcal{A}$ outputs a valid forgery for the signcryption scheme $\mathcal{SC}$. Then*

$$\Pr[\mathsf{Forge}] \leq (n+1)\,\mathsf{Adv}^{\mathsf{Forge}}_{\mathcal{SC}}(t).$$

Proof. We suppose the adversary $\mathcal{A}$ gains her advantage by forging the signature (within ciphertext) of the group leader U_ρ. The forger $\mathcal{F}$ chooses a random $U_d \in \mathcal{U}$ with identity ID_d as its guess for the leader U_ρ and as the user for which he wishes to forge a signcrypted ciphertext in the EUF-IBSC-CMA security game. $\mathcal{F}$ honestly generates the public/private key pairs for each user with identity different from ID_d using the Extract oracle and, using these keys and the Sign/Encrypt and Decrypt/Verify oracles, simulates the group key oracle queries of $\mathcal{A}$ in the usual way. If at any point, $\mathcal{A}$ issues the Corrupt query on ID_d, then $\mathcal{F}$'s guess for the value of ρ was incorrect and $\mathcal{F}$ aborts the simulation. However, if $\mathcal{A}$ outputs a new ciphertext c that decrypts to a valid message-signature pair for U_d, resulting in the event Forge, $\mathcal{F}$ returns the ciphertext.

Suppose that $\mathcal{A}$ succeeds in forging a signature within ciphertext with probability $\Pr[\mathsf{Forge}]$. The probability that this is a forgery for the user U_d is $1/(n+1)$. Thus, the probability of success for $\mathcal{F}$ is given by $\mathsf{Adv}^{\mathsf{Forge}}_{\mathcal{F},\mathcal{SC}}(t) = \frac{1}{n+1}\Pr[\mathsf{Forge}]$, and since $\mathsf{Adv}^{\mathsf{Forge}}_{\mathcal{F},\mathcal{SC}}(t) \leq \mathsf{Adv}^{\mathsf{Forge}}_{\mathcal{SC}}(t)$, we have $\Pr[\mathsf{Forge}] \leq (n+1)\,\mathsf{Adv}^{\mathsf{Forge}}_{\mathcal{SC}}(t)$.

Indistinguishability Attack on $\mathcal{SC}$. Now suppose that an adversary $\mathcal{A}$ gains her advantage without forging the protocol transcripts. We use $\mathcal{A}$ to build a distinguishing algorithm $\mathcal{D}$ for the signcryption scheme $\mathcal{SC}$ that has non-negligible advantage in the IND-IBSC-CCA2 security game.

Lemma 2. *Let* Distinguish *be the event that an adversary $\mathcal{A}$ can distinguish between ciphertexts of the signcryption scheme $\mathcal{SC}$. Then*

$$\Pr[\mathsf{Distinguish}] \leq (n+1)\,q_{ex}\mathsf{Adv}^{\mathsf{Distinguish}}_{\mathcal{SC}}(t),$$

where q_{ex} is the number of Execute *queries issued by $\mathcal{A}$.*

Proof. We suppose the adversary $\mathcal{A}$ gains her advantage by distinguishing between the signcrypted ciphertexts sent by the group leader U_ρ. The algorithm $\mathcal{D}$ chooses a random $U_d \in \mathcal{U}$ with identity ID_d as its guess for the group leader U_ρ and the sender of the signcrypted ciphertext in the IND-IBSC-CCA2 security game, and sets the remaining members of $\mathcal{U}$ as the receivers. The distinguisher $\mathcal{D}$ chooses $m_0, m_1 \in_R \{0,1\}^*$ and outputs the sender identity ID_d, the set of receiver identities $\{\mathsf{ID}_i\}_{i \neq d}$ and the messages $\{m_0, m_1\}$ in IND-IBSC-CCA2 security game. The challenger responds with the challenge signcryption ciphertext $c^* = \mathsf{Sign/Encrypt}(m_\phi, \mathsf{ID}_d, \{\mathsf{ID}_i\})$, which has the form $\langle \gamma_1, ..., \gamma_n \rangle$ since the plaintext message is signcrypted for each individual receiver. We suppose that $\mathcal{A}$ makes a maximum of q_{ex} $\mathsf{Execute}$ queries, where q_{ex} is polynomial in the security parameter ℓ. $\mathcal{D}$ chooses a random session identifier $\beta \in [1, q_{ex}]$ for which he will give the adversary the challenge ciphertext c^*.

The algorithm $\mathcal{D}$ models the hash functions H_1 and H_2 as random oracles. To be consistent amongst the adversary's queries, $\mathcal{D}$ maintains the following lists, where α refers to the session number:

- List L_1 records entries of the form $(\alpha, X_1, ..., X_n, h_1)$ to maintain consistency in oracle H_1 queries, where $h_1 = H_1(X_1, ..., X_n)$.
- List L_2 records entries of the form $(\alpha, \mathcal{U}, session_{ID}, N, h_2)$ to maintain consistency in oracle H_2 queries, where $h_2 = H_2(\text{“}\mathcal{U} \parallel session_{ID} \parallel N\text{”})$.
- List L_{AGKA} records entries of the form

  $$(\alpha, \mathcal{U}, session_{ID}, ID_{\alpha_1}, \sigma_{\alpha_1}, ..., ID_{\alpha_n}, \sigma_{\alpha_n}, ID_{\alpha_0}, N, \sigma, c_{\alpha_1}, ..., c_{\alpha_n}, h_1, h_2, K)$$

  to maintain consistency in Send, $\mathsf{Execute}$ and Reveal queries. This list is essentially a transcript of the α^{th} session of the protocol.

We denote the α^{th} entry of a list by $L(\alpha)$. Using these lists, the algorithm $\mathcal{D}$ responds to $\mathcal{A}$'s queries as follows:

$\mathsf{Send}(\Pi_U^\alpha, M)$] Algorithm $\mathcal{D}$ handles Send queries in the following cases:

1. If $M =$“`init` $\parallel$ `leader`”, so that the message is to initiate the protocol with U as the leader, we have the following three subcases:
   (a) If $U = U_d$ and $\alpha = \beta$, then $\mathcal{D}$ returns the signcrypted values $\langle \gamma_1, ..., \gamma_n \rangle$ in place of the leader's signcrypted contribution values $\langle c_{\alpha_1}, ..., c_{\alpha_n} \rangle$.
   (b) If $U \neq U_d$ and $\alpha = \beta$, then $\mathcal{D}$'s choice for the group leader was incorrect and the algorithm fails.
   (c) Otherwise, $\mathcal{D}$ chooses a random $N \in \{0,1\}^*$ and responds with the ciphertexts $\langle c_{\alpha_1}, ..., c_{\alpha_n} \rangle$ obtained from $\mathsf{Sign/Encrypt}(\text{“}\mathcal{U} \parallel session_{ID} \parallel N\text{”}, \mathsf{ID}_U, \{\mathsf{ID}_{\alpha_1}, ..., \mathsf{ID}_{\alpha_n}\})$. The algorithm records $(\mathcal{U}, session_{ID}, \mathsf{ID}_U, N, c_{\alpha_1}, ..., c_{\alpha_n})$ in $L_{AGKA}(\alpha)$.
2. If $M =$“`init` $\parallel$ `responder`”, so that the message is to initiate the protocol with U as the responder, we have the following two subcases:
   (a) If $U = U_d$ and $\alpha = \beta$, then $\mathcal{D}$'s choice for the group leader was incorrect and the algorithm fails.

(b) Otherwise, $\mathcal{D}$ runs the protocol normally as a responder. We note that the security game does not give $\mathcal{D}$ access to the signature scheme $\mathcal{S}$. However, we can still sign responder contributions due to the $\mathcal{S}$-verifiability of the signcryption scheme. The algorithm queries $\mathsf{Sign/Encrypt}(\text{“}\mathcal{U} \parallel session_{ID}\text{”}, \mathsf{ID}_U, \mathsf{ID}'_i)$ for some identity ID'_i to obtain a ciphertext c' and subsequently queries $\mathsf{Decrypt/Verify}(c', \mathsf{ID}_U, \mathsf{ID}'_i)$ to obtain $(\text{“}\mathcal{U} \parallel session_{ID}\text{”}, \sigma_U)$, where σ_U is a valid signature on “$\mathcal{U} \parallel session_{ID}$” for the responder U. We note that there is no restriction on queries of this form in the IND-IBSC-CCA2 security game. $\mathcal{D}$ returns the signature and records $(\mathcal{U}, session_{ID}, ID_U, \sigma_U)$ in $L_{AGKA}(\alpha)$.

3. If M is not to initiate the protocol, then Π_U^α accepts the message M and responds as follows. If M represents the signed contribution of a responder, then $\mathcal{D}$ records (ID_U, σ_U) in $L_{AGKA}(\alpha)$. In addition, we have the following subcases:

   (a) If U is the group leader and M represents the last responder contribution σ_{α_i} for session α, then $\mathcal{D}$ verifies the signatures of all responders and accepts the session key if the verification holds. $\mathcal{D}$ outputs the outcome of acceptance.

   (b) If U is a responder and M represents the set of signcryption ciphertexts $(c_{\alpha_1}, ..., c_{\alpha_n})$ of the leader for session α, then $\mathcal{D}$ accepts the information, provided it is in the expected format. As long as $\alpha \neq \beta$, $\mathcal{D}$ obtains the valid message-signature pair $(\text{“}\mathcal{U} \parallel session_{ID} \parallel N\text{”}, \sigma)$ by querying $\mathsf{Decrypt/Verify}(c_{\alpha_i}, \mathsf{ID}_{\alpha_i}, \mathsf{ID}_U)$. Algorithm $\mathcal{D}$ verifies all responder signatures and the detachable signature, outputs the outcome of acceptance and records $(\mathcal{U}, session_{ID}, \mathsf{ID}_U, N, \sigma, c_{\alpha_1}, ..., c_{\alpha_n})$ in $L_{AGKA}(\alpha)$.

$\mathsf{Execute}(U_{i_0}, U_{i_1}, ..., U_{i_n})$ $\mathcal{D}$ executes the protocol for the leader U_{i_0} and the responders $\mathcal{R} = \{U_{i_1}, ..., U_{i_n}\}$, according to the steps outlined in the Send query.

$\mathsf{Reveal}(\Pi_U^\alpha)$ We first assume that oracle Π_U^α has accepted the session key and thus, Π_U^α has collected all responder contributions $\{X_{\alpha_1}, ..., X_{\alpha_n}\}$ and the leader contribution N. The session keys are given by the bitwise addition of the outputs h_1 and h_2 of the random oracles H_1 and H_2, respectively. If the key for session α has not yet been revealed, $\mathcal{D}$ obtains h_1 and h_2 from $L_{AGKA}(\alpha)$ if they exist, or queries H_1 and H_2 otherwise. $\mathcal{D}$ returns $K = h_1 \oplus h_2$ as the α^{th} session key and adds K to $L_{AGKA}(\alpha)$. If the key for session α has been revealed before, $\mathcal{D}$ returns K from $L_{AGKA}(\alpha)$.

$\mathsf{Corrupt}(U)$ If U is the group leader, $\mathcal{D}$ responds with the private key obtained from the $\mathsf{Extract}$ query. Otherwise, $\mathcal{D}$ aborts the simulation.

$\mathsf{Test}(\Pi_U^\alpha)$ If $\alpha = \beta$ and the group leader of the α^{th} session is U_d, then $\mathcal{D}$ returns a random string as the session key. Otherwise, the algorithm fails.

At some point during the game, $\mathcal{A}$ will output a guess for the value of b. Recall that $\mathcal{A}$ queries the hash functions H_1 and H_2 to compute the session key. Since these hash functions are modelled as random oracles, $\mathcal{A}$ has no advantage in guessing session keys that were not amongst these oracle queries. The algorithm $\mathcal{D}$ examines all queries $H_2(\mathcal{U} \parallel session_{ID} \parallel N)$ made by $\mathcal{A}$ to the oracle H_2,

which $\mathcal{D}$ has recorded in the list L_2. If at any point, $\mathcal{D}$ finds a query with $N = m_0$, then it outputs $\phi = 0$ as its guess. Otherwise, $\mathcal{D}$ returns $\phi = 1$.

Suppose that $\mathcal{A}$ distinguishes between the ciphertexts with probability $\Pr[\mathsf{Distinguish}]$. The probability that the simulation does not fail (i.e. $\alpha = \beta$ and $U_\rho = U_d$) is $1/(n+1)\,q_{ex}$. Thus, the probability of success for $\mathcal{D}$ is given by $\mathsf{Adv}^{\mathsf{Distinguish}}_{\mathcal{D},\mathcal{SC}}(t) = \frac{1}{(n+1)q_{ex}} \Pr[\mathsf{Distinguish}]$, and since $\mathsf{Adv}^{\mathsf{Distinguish}}_{\mathcal{D},\mathcal{SC}}(t) \leq \mathsf{Adv}^{\mathsf{Distinguish}}_{\mathcal{SC}}(t)$, then $\Pr[\mathsf{Distinguish}] \leq (n+1)\,q_{ex}\mathsf{Adv}^{\mathsf{Distinguish}}_{\mathcal{SC}}(t)$.

This completes the proof for theorem 1 for both possible cases.

6 An Example of the COMPASS Protocol

In this section, we provide a concrete example of the COMPASS protocol using the signcryption scheme of [24], which we call the YYHZ signcryption scheme. As discussed in Section 3, the YYHZ scheme is an efficient multi-receiver variant of the identity-based signcryption scheme of [13]. Furthermore, the YYHZ signcryption scheme provides $\mathcal{S}$-verifiability using the identity-based signature scheme of [12], which produces multipurpose signatures and was shown to support efficient batch verification by [17]. As a result, the YYHZ signcryption scheme is an optimal candidate for the COMPASS protocol.

[22] showed that the proofs for confidentiality and authentication in [24] were flawed and proposed a slightly modified version that is IND-IBSC-CCA2 and EUF-IBSC-CMA secure in the multi-receiver signcryption model. Consequently, we use the modified version from [22] in the YYHZ-COMPASS protocol.

6.1 Setup Phase

The YYHZ signature scheme uses an ID-PKI similar to that of the IBE scheme of [5]. The ID-PKI relies on a special mathematical primitive known as a bilinear pairing. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be cyclic groups of large prime order q, where we take $\mathbb{G}_1$ to be a subgroup of points on an elliptic curve and $\mathbb{G}_2$ to be a subgroup of the multiplicative group of a finite field. A pairing is a non-degenerate map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ that satisfies the property that $e(aP, bQ) = e(P, Q)^{ab}$ for all $a, b \in \mathbb{Z}_q^\times$ and $P, Q \in \mathbb{G}_1$. For more information, we refer the reader to [5].

The PKG uses the **Setup** and **Extract** algorithms of the ID-PKI to issue a public/private key pair (Q_i, d_i) to each group member $U_i \in \mathcal{U}$. The algorithms are defined as follows:

- Setup: Given a security parameter ℓ, the PKG generates the public system parameters

$$\mathsf{params} = \langle \mathbb{G}_1, \mathbb{G}_2, e, q, P, P_{pub}, R, \theta, H_0, H_1, H_2, H_3, H_4, H_5 \rangle .$$

  where P is a generator of $\mathbb{G}_1$, $R \in_R \mathbb{G}_1^\times$ and $\{H_i\}$ are cryptographic hash functions, to be defined in the protocol. The PKG chooses a master secret key $s \in_R \mathbb{Z}_q^\times$ and computes the global public key $P_{pub} = sP$ and the public value $\theta = e(R, P_{pub})$.

– Extract: Given an identity ID_i, the PKG returns the public key $Q_i = H_0(\mathsf{ID}_i)$, using $H_0 : \{0,1\}^* \to \mathbb{G}_1^\times$, and the private key $d_i = sQ_i$.

6.2 The YYHZ-COMPASS Protocol

To begin group communication, each member obtains the session identifier $session_{ID} \in \{0,1\}^*$, which may be taken from some preexisting list. The YYHZ-COMPASS protocol is executed in a single round as follows:

Round 1 (Responders). Using the Cha-Cheon signature scheme, each responder $U_i \in \mathcal{R}$ signs the session string "$\mathcal{U} \parallel session_{ID}$" by

1. Choosing $r_i \in_R \mathbb{Z}_q^\times$.
2. Computing
   – $X_i = r_i Q_i$,
   – $h_i = H_1(X_i, \text{"}\mathcal{U} \parallel session_{ID}\text{"})$ using $H_1 : \mathbb{G}_1 \times \{0,1\}^* \to \mathbb{Z}_q^\times$,
   – $Z_i = (r_i + h_i)d_i$.
3. Broadcasting the signature $\sigma_i = (X_i, Z_i)$ to the rest of the group.

Round 1 (Group Leader). Meanwhile, the group leader U_0 chooses a secret key contribution $N \in_R \{0,1\}^*$ and uses the YYHZ signcryption scheme to signcrypt her contribution for the group of responders by

1. Choosing $r_0, k_0 \in_R \mathbb{Z}_q^\times$.
2. Computing
   – $T = k_0 P$,
   – $X_0 = r_0 Q_0$,
   – $h_0 = H_2\left(T, X_0, \text{"}\mathsf{ID}_0 \parallel \mathcal{U} \parallel session_{ID} \parallel N\text{"}\right)$ using $H_2 : \mathbb{G}_1^2 \times \{0,1\}^* \to \mathbb{Z}_q^\times$,
   – $Z_0 = (r_0 + h_0)d_0$,
   – $\omega = e(Z_0, P)$,
   – $y = (\text{"}\mathcal{U} \parallel session_{ID} \parallel N\text{"} \parallel \sigma_0) \oplus H_3(\omega)$, using $H_3 : \mathbb{G}_2 \to \{0,1\}^*$, where $\sigma_0 = (X_0, Z_0)$,
   – $W = \omega \cdot \theta^{k_0}$,
   – $c_i = k_0\left(Q_i + R\right)$, for each responder $U_i \in \mathcal{R}$.
3. Broadcasting the ciphertext set $\langle y, T, W, c_1, c_2, ..., c_n \rangle$, along with some list $\mathcal{L}$ indicating which ciphertext c_i corresponds to which responder U_i.

Key Computation. Upon receiving the ciphertext c_i from the group leader, each responder U_i computes the session key by

1. Computing
   – $\omega' = W \cdot e(d_i, T) \cdot e(P_{pub}, c_i)^{-1}$,
   – $\text{"}\mathcal{U} \parallel session_{ID} \parallel N\text{"} \parallel \sigma_0 = y \oplus H_3\left(\omega'\right)$, [2]

[2] We note that each responder may verify that $\omega' = e(Z_0, P)$ to be sure that they are in possession of the correct value ω, although this computation is not necessary.

Table 3. Computational costs compared with the LKKR and CHL protocols

Protocol	YYHZ-COMPASS		LKKR	CHL
	Leader u_0	Responder u_i		
$\mathcal{P}$	3	4	–	–
Total	$4n+3$		$5hn+5h$	$6n+6$
$\mathcal{M}$	$4n+6$	$3n+5$	–	–
Total	$3n^2+9n+6$		$\frac{4h+9}{2}n+2h$	$n^2+11n+10$

to recover the group leader's secret contribution N and the detachable signature $\sigma_0 = (X_0, Z_0)$.

2. Choosing $(\delta_0, \delta_1, ..., \delta_n) \in_R \left(\mathbb{Z}_q^{\times}\right)^{n+1}$.
3. Batch verifying the set of signatures $\{\sigma_0, \sigma_1, ..., \sigma_n\}$ by checking that

$$e(\sum_{i=0}^{n} \delta_i Z_i, Q) = e(\sum_{i=0}^{n} \delta_i \left(X_i + h_i Q_i\right), P_{pub}),$$

as in [17].

4. If the verification holds, they compute the session key as

$$K = H_4(X_1, ..., X_n) \oplus H_5(\text{“}\mathcal{U} \parallel session_{ID} \parallel N\text{”}),$$

using the hash functions $H_4 : \mathbb{G}_1^n \to \{0,1\}^*$ and $H_5 : \{0,1\}^* \to \{0,1\}^*$.

6.3 Efficiency of the YYHZ-COMPASS Protocol

We compare the computational costs of the YYHZ-COMPASS protocol with two of the more prominent pairing-based GKA protocols from the literature: the LKKR protocol of [20], based on the TGDH protocol from [19], and the CHL protocol of [14], based on the BD protocol from [10]. Since the LKKR protocol does not provide authentication, we include the costs incurred by using the authenticated tripartite key exchange from [25]. To ease in our comparison, we assume that $n+1 = 3^h$ for some $h \in \mathbb{Z}^+$, so that the LKKR key tree is a perfect ternary tree of height h. We present the costs of the modified CHL protocol proposed by the authors in [15], so that it is protected against insider impersonation attacks.

We compare the protocols in terms of total pairing computations $\mathcal{P}$ and total scalar multiplications $\mathcal{M}$ in the elliptic curve group $\mathbb{G}_1$. We also provide the individual computational costs of the leader and responder for the YYHZ-COMPASS protocol. As shown in Table 3, the YYHZ-COMPASS protocol provides huge computational gains over the LKKR and CHL protocols in terms of the more expensive pairing operation.

7 Conclusions and Future Work

In this paper, we proposed a computationally asymmetric authenticated group key agreement protocol from identity-based signcryption schemes, which we call

the COMPASS protocol, that achieves the lower bound for communication complexity derived by [4]. By choosing a signcryption scheme that satisfies some desirable criteria, we ensure computational and communication efficiency, while also providing the strongest contributory property of key integrity from [2]. We provided a formal proof reducing the semantic security of the protocol to the security of the underlying signcryption scheme in the random oracle model and provided a concrete example of the COMPASS protocol using the multi-receiver signcryption scheme from [24], which is more computationally efficient than two of the more well-known pairing-based AGKA protocols from the literature.

While we illustrated the COMPASS protocol using the signcryption scheme from [24], it would be interesting to consider other identity-based signcryption schemes from current literature, such as the efficient $\mathcal{S}$-verifiable signcryption scheme of [3]. To produce an efficient group key agreement protocol, their signcryption scheme must be modified to support multiple recipients and provide batch verification of signatures in the underlying signature scheme, while preserving the provable security of the scheme. Can we construct new identity-based signcryption schemes that yield even more efficient COMPASS protocols?

References

1. An, J.H., Dodis, Y., Rabin, T.: On the Security of Joint Signature and Encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 83–107. Springer, Heidelberg (2002)
2. Ateniese, G., Steiner, M., Tsudik, G.: Authenticated Group Key Agreement and Friends. In: CCS 1998: Proceedings of the 5th ACM Conference on Computer and Communications Security, pp. 17–26. ACM Press, New York (1998)
3. Barreto, P.S.L.M., Libert, B., McCullagh, N., Quisquater, J.-J.: Efficient and Provably-Secure Identity-Based Signatures and Signcryption from Bilinear Maps. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 515–532. Springer, Heidelberg (2005)
4. Becker, K., Wille, U.: Communication Complexity of Group Key Distribution. In: CCS 1998: Proceedings of the 5th ACM Conference on Computer and Communications Security, pp. 1–6. ACM Press, New York (1998)
5. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
6. Boyd, C.: On Key Agreement and Conference Key Agreement. In: Mu, Y., Pieprzyk, J.P., Varadharajan, V. (eds.) ACISP 1997. LNCS, vol. 1270, pp. 294–302. Springer, Heidelberg (1997)
7. Boyd, C., González Nieto, J.M.: Round-Optimal Contributory Conference Key Agreement. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 161–174. Springer, Heidelberg (2002)
8. Boyen, X.: Multipurpose Identity-Based Signcryption – A Swiss Army Knife for Identity-Based Cryptography. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 383–399. Springer, Heidelberg (2003)
9. Bresson, E., Chevassut, O., Pointcheval, D., Quisquater, J.-J.: Provably Authenticated Group Diffie-Hellman Key Exchange. In: CCS 2001: Proceedings of the 8th ACM Conference on Computer and Communications Security, pp. 255–264. ACM, New York (2001)

10. Burmester, M., Desmedt, Y.: A Secure and Efficient Conference Key Distribution System. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 275–286. Springer, Heidelberg (1995)
11. Cao, X., Kou, W., Du, X.: A pairing-free identity-based authenticated key agreement protocol with minimal message exchanges. Information Science 180, 2895–2903 (2010)
12. Cha, J.C., Cheon, J.H.: An Identity-Based Signature from Gap Diffie-Hellman Groups. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 18–30. Springer, Heidelberg (2002)
13. Chen, L., Malone-Lee, J.: Improved Identity-Based Signcryption. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 362–379. Springer, Heidelberg (2005)
14. Choi, K.Y., Hwang, J.Y., Lee, D.H.: Efficient ID-based Group Key Agreement with Bilinear Maps. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 130–144. Springer, Heidelberg (2004)
15. Choi, K.Y., Hwang, J.Y., Lee, D.H.: ID-Based Authenticated Group Key Agreement Secure Against Insider Attacks. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 91, 1828–1830 (2008)
16. Choo, K.R.: Key Establishment: Proofs and Refutations, PhD, Queensland University of Technology, Brisbane, Australia (2006)
17. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical Short Signature Batch Verification. Cryptology ePrint Archive, Report 2008/015 (2008)
18. Guo, H., Li, Z., Mu, Y., Zhang, X.: Provably secure identity-based authenticated key agreement protocols with malicious private key generators. Information Science 181, 628–647 (2011)
19. Kim, Y., Perrig, A., Tsudik, G.: Tree-Based Group Key Agreement. ACM Transactions on Information and System Security 7, 60–96 (2004)
20. Lee, S., Kim, Y., Kim, K., Ryu, D.-H.: An Efficient Tree-Based Group Key Agreement Using Bilinear Map. In: Zhou, J., Yung, M., Han, Y. (eds.) ACNS 2003. LNCS, vol. 2846, pp. 357–371. Springer, Heidelberg (2003)
21. Shamir, A.: Identity-Based Cryptosystems and Signature Schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
22. Sharmila Deva Selvi, S., Sree Vivek, S., Srinivasan, R., Pandu Rangan, C.: An Efficient Identity-Based Signcryption Scheme for Multiple Receivers. In: Takagi, T., Mambo, M. (eds.) IWSEC 2009. LNCS, vol. 5824, pp. 71–88. Springer, Heidelberg (2009)
23. Shin, J.-B., Lee, K., Shim, K.: New DSA-Verifiable Signcryption Schemes. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 35–47. Springer, Heidelberg (2003)
24. Yu, Y., Yang, B., Huang, X., Zhang, M.: Efficient Identity-Based Signcryption Scheme for Multiple Receivers. In: Xiao, B., Yang, L.T., Ma, J., Muller-Schloer, C., Hua, Y. (eds.) ATC 2007. LNCS, vol. 4610, pp. 13–21. Springer, Heidelberg (2007)
25. Zhang, F., Liu, S., Kim, K.: ID-Based One Round Authenticated Tripartite Key Agreement Protocol with Pairings. Cryptology ePrint Archive, Report 2002/122 (2002)

26. Zhang, L., Wu, Q., Qin, B., Domingo-Ferrer, J.: Provably secure one-round identity-based authenticated asymmetric group key agreement protocol. Information Science 181, 4318–4329 (2011)
27. Zheng, Y.: Digital Signcryption or How to Achieve Cost (Signature & Encryption) << Cost(Signature) + Cost(Encryption). In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 165–179. Springer, Heidelberg (1997)
28. Zheng, Y., Imai, H.: Compact and Unforgeable Key Establishment over an ATM Network. In: Proceedings of IEEE INFOCOM 1998, vol. 2, pp. 411–418. IEEE, Inc. (1998)

Classifying Online Social Network Users through the Social Graph*

Cristina Pérez-Solà[1] and Jordi Herrera-Joancomartí[1,2]

[1] Dept. d'Enginyeria de la Informació i les Comunicacions
Universitat Autònoma de Barcelona
08193 Bellaterra, Catalonia, Spain
{cperez,jherrera}@deic.uab.cat
[2] Internet Interdisciplinary Institute (IN3) - UOC

Abstract. In this paper, we address the problem of classifying online social network users using a naively anonymized version of a social graph. We use two main user attributes defined by the graph structure to build an initial classifier, node degree and clustering coefficient, and then exploit user relationships to build a second classifier. We describe how to combine these two classifiers to build an Online Social Network (OSN) user classifier and then we evaluate the performance of our architecture by trying to solve two different classification problems (a binary and a multiclass problem) using data extracted from Twitter. Results show that the proposed classifier is sound and that both classification problems are feasible to solve by an attacker who is able to obtain a naively anonymized version of the social graph.

Keywords: Online Social Networks, Relational Classifiers, Graph Anonymization.

1 Introduction

Online Social Networks (OSN) are web services that allow users to create a public (or partially public) profile describing some information about themselves and share information with other users of the network [1]. Their most characteristic feature is that they allow users to create explicit relationships between them in the network. Graphs that are used to represent these explicit links are called social graphs and they have been widely used to analyze OSN in a broad variety of studies. A social graph is defined as a graph where nodes represent users in an OSN and edges denote explicit links between them. Node attributes are then information about the user (such as age, gender, or sexual preferences).

Recent studies report that high homophily is observed on OSN user communications. For instance, it has been shown in [2] that attention is homophilous within Twitter elite users, with users listening to what other similar users have

* This work was partially supported by the Spanish MCYT and the FEDER funds under grants TSI2007-65406-C03-03 "E-AEGIS", TIN2010-15764 "N-KHRONOUS", and CONSOLIDER CSD2007-00004 "ARES".

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 115–131, 2013.

to say in the network. Since users pay attention to other similar users, it seems reasonable to think that it is possible to use the information about who is linked to whom in an OSN to classify these OSN users, even when attributes describing those users have been removed from the graph.

Classification is one of the basic techniques in data mining processes. Given a set of labeled samples, the goal is to assign labels to the rest of the dataset. By doing so, it is possible to learn desired properties of those samples. It is immediate to see that depending on the nature of these inferred properties, user's privacy may be compromised with the classification process.

User classification is far from being an innoxious process. On one hand, if the categories used on the process correspond to sensitive attributes, classification can directly lead to private attribute disclosure. One of the most famous examples of this specific problem was reported in [3], where the authors were able to predict the sexual orientation of Facebook users that did not have that information on their profile by using information on their friend's profiles. On the other hand, a user may not be concerned about the attribute disclosure per se but about the consequences that this disclosure may have. For instance, being classified in a specific group may be the difference between getting a mortgage loan denied or approved. In any case, classification results in the disclosure of user attributes that the user did not explicitly approve. Since the user is not able to control the disclosure of the information about himself anymore[1], a privacy breach occurs.

In this paper, we address the problem of classifying OSN users using a naively anonymized copy of a social graph. In order to create a naively anonymized copy, all identifying attributes are usually removed from the original graph. In our case, we go one step beyond and remove all identifiers and node attributes, so that no semantic information can be used in the classification process. Note that since no edges have been added nor deleted from the original graph, the anonymized copy has exactly the same structure than the original graph.

We assume that the adversary, who wants to classify OSN users, is able to learn labels for some subset of the nodes of the graph a priori. These nodes, for which the attacker is able to obtain labels beforehand, are used as training samples in the classification process. Then, the adversary infers labels for the rest of the dataset, learning information from users for which no prior information could be found.

The main contribution of this paper is to demonstrate that the graph structure alone is enough to classify OSN users and, therefore, a naive anonymization technique that removes node attributes is not enough to protect users privacy. To prove our claim, we use the Twitter network as a testbed and we classify Twitter users into two different sets of categories. Initially, we classify users as either individual users or companies. We refer to this classification as the binary classification problem. After that, we move on to a more detailed classification,

[1] Here we are referring to Westin's privacy definition [4] which states that *Privacy is the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others.*

allocating users into five different categories: bloggers, celebrities, media, organizations or non-elite users. We designate this classification as the multiclass classification problem. Results show that even with the only information of the graph structure, users can be classified up to a certain accuracy level.

The rest of the paper is organized as follows. Section 2 reviews the state of the art. In Section 3, we describe the Twitter network, the data collection process, and its representation. Section 4 analyses the information contained in the Twitter social graph. Then, Section 5 discuses the architecture proposed for the classification process. In Section 6 we detail the experimental results obtained using the classifier presented in Section 5. Finally, Section 7 provides the conclusions and gives some guidelines for further research.

2 State of the Art

In contrast with traditional non-relational data, networked data is characterized for containing entities and some kind of relationships between them. Nowadays, OSN are one of the most popular examples of networked data, containing information about users and their interaction.

The problem of classifying networked data has been a recent focus of activity in the machine learning research community, with special interest on adapting traditional machine learning techniques to networked data classification.

In [5], the authors present a relational classifier toolkit. Beyond the actual toolkit itself and by describing each of its modules, the authors review different algorithms that can be used to classify networked data.

Many algorithms for relational classifiers have been proposed in the past. In [6] the authors present the Relational Neighbor (RL) classifier based on the principle of homophily, where the probability of a sample belonging to a given class is proportional to the number of neighbors of that sample belonging to the same class. The Weighted Vote Relational Classifier (WVRN) estimates class-membership of a node as the weighted mean of the class-membership probabilities of the neighbors of that node. The Class-Distribution Relational Neighbor classifier (CDRN) is presented in [5], where the probability of class membership of a node is estimated by the similarity of its class vector with the class reference vector. The class vector of a node is defined as the vector of summed linkage weights to the various classes and the class reference vector for a given class is the average of the class vectors for nodes known to be of that class. Network Only Bayes classifier (nBC) [7] uses naive Bayes classification based on the classes of the nodes' neighbors to classify hyperlinked documents. In [8] the Network-Only Link-Based classification (nLB) is presented, which uses regularized logistic regression models to classify networked data.

Collective inference may improve probabilistic inference in networked data [9]. Many collective inference methods are used in relational learning: Gibbs sampling [10], relaxation labeling [7], and iterative classification [8,11] are the most used.

Experimental methodologies for both within-network and between network classification are reviewed in [12], where the approaches of different authors to the problem are summarized.

Relational classification has been applied to email classification [13], with a dataset of mails being linked only by parent-children relationships; to topic classification of hypertext documents [7]; to predict movie success with IMDBb data, linking movies with a shared production company [6,5]; to sub-topic prediction in machine learning papers [5]; to age, gender, and location prediction of bloggers [14]; and many other networked data classification problems.

Networked data has also been studied from a privacy preservation point of view. Several methods have been proposed to anonymize networked data, although most of them are not suitable for large scale graphs. Moreover, it has been shown that anonymizing networked data while maintaining its utility is also difficult. Far from being solved, the problem of anonymizing networked data is open and still challenging.

The first step for anonymizing networked data is to remove identifying attributes. Usually, random identifiers are then included. This procedure is known as *naive anonymization* [15].

Some anonymization techniques include graph perturbation methods, where edges and/or nodes are added and deleted to modify the graph structure. *Random perturbation* [15] consist on randomly deleting m edges of the initial graph, and afterwards randomly adding m new edges. In [16], the authors propose several strategies for anonymizing graphs which are combined with traditional tabular data anonymization techniques. These strategies add and/or delete edges of the graph taking into account different criteria instead of randomly selecting the modified edges. Some other techniques try to adapt the k-anonymity paradigm for tabular data to networked data. Depending on their definition for a k-anonymous graph, several proposals appear: in [17], the authors modify the graph so that there is at least k nodes with the same degree; in [18], the authors assume that the attacker knows the neighborhood at distance 1 of a node and they say that a graph is k-anonymous if at least k different nodes have the same neighborhood graph (or an isomorphism of that graph); other variants can be found in [19] and [20]. Although all these techniques offer better anonymization than the naive approach, the utility of the released graph is usually affected. For this reason, naive anonymization is still used to anonymize graph data before releasing it.

3 The Twitter Network

Twitter is a famous microblogging service that allows users to publish messages up to 140 characters. Twitter has gained popularity as an almost real-time source of information and as a platform for organizing masses.

Users messages in the Twitter network are called tweets. Users can subscribe to other users' updates so that they receive all their tweets, establishing in this way topological links between users. These relationships are not bidirectional,

so Alice can be following Bob's updates and Bob may not be following Alice's updates at all.

Twitter is used with many different purposes and, because of that, many different uses are given to each Twitter account. While behind some of these accounts there is only a single non-famous person who comments on his topics of interest, entire multinationals can be found backing on other accounts. Even some of the news media companies have their own Twitter account. This diversity of users is both enriching and a challenge for anyone who deals with Twitter data, from its own engineers to advertisers or external data analysts.

In 2009, Twitter introduced a new feature in their network: the Twitter lists. These lists allow users to create lists of Twitter accounts, so that it is possible to organize both followed and not-followed users. Each Twitter list has its own view that shows a stream of tweets from all the users included in that list. By doing so, users can get an aggregated overview on what is going on on that list. Moreover, once a list has been created, any other user of the network can subscribe to that list. This feature considerably increases the functionality of Twitter lists by allowing people to use the lists of other users to enhance their experience in the network.

3.1 Obtaining Twitter Data

Twitter allows developers to access its stored data via Twitter APIs. We used this feature in order to obtain, on one hand, information about user's relationships and, on the other hand, list membership data.

To obtain users relationships, we start by selecting an initial node which will be used as the seed for our crawl. After that, we fetch the list of followers of that initial user and proceed with the crawling in a Breadth First Search manner[2]. Social graphs are then created from this information in an almost immediate way. Users obtained from the crawling process are mapped to nodes of the social graph and edges are placed within users that explicitly create a relationship in the network. Since Twitter links are directed, we use a directed graph to represent those social graphs.

After obtaining users and their relationships, we proceed to obtain list membership information, discovering in which lists does every of the previously crawled users appear. For each list, name, slug, and description are retrieved, as well as their subscribers and members count, and ownership information.

3.2 Twitter Data Representation

Social networks are usually visualized as graphs, where nodes represent users and edges describe relationships among them. Having this identification in mind, we use $G = (V, E)$ to describe our Twitter social graph. The set $V = \{v_i, \text{ for } i = 1, \cdots, n\}$ contains the nodes of the graph that are identified as users of the Twitter network. On the other hand, E is the set of edges, ordered pairs of

[2] For a detailed description of the BFS algorithm, the readers can refer to [21].

different elements of V. As already mentioned, edges in social graphs represent relationships among users and, in our case, the edge (v_i, v_j) represents that user v_i follows user v_j in the Twitter network. Notice that since Twitter relationships are not bidirectional (as in Facebook), the corresponding graph is directed and even if v_i follows v_j ($(v_i, v_j) \in E$), it does not imply that user v_j also follows user v_i ($(v_j, v_i) \in E$).

In an undirected graph, we denote by $\Gamma(v_i)$ the set of adjacent nodes of v_i and the node degree is the cardinality of such set, $deg(v_i) = |\Gamma(v_i)|$. However, in a directed graph, we must distinguish between successors $\Gamma(v_i)$ and predecessors $\Gamma^{-1}(v_i)$ of a node v_i. The set of successors of v_i is defined as $\Gamma(v_i) = \{v_j \in V$ s.t. $\exists\ e \in E$ with $e = (v_i, v_j)\}$. In a similar fashion, the set of predecessors of v_i can be defined as $\Gamma^{-1}(v_i) = \{v_j \in V$ s.t. $\exists\ e \in E$ with $e = (v_j, v_i)\}$. Both definitions provide the concept of outdegree and indegree of a node v_i that can be formally defined as $|\Gamma(v_i)|$ and $|\Gamma^{-1}(v_i)|$, respectively.

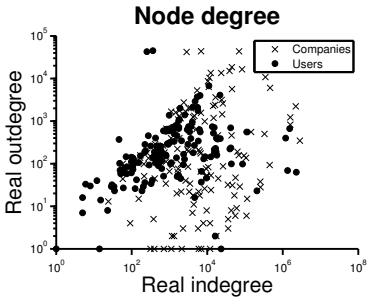
4 Network Information in the Twitter Dataset

In order to classify Twitter users based on the network topology, first we have to check if there really exist information in the network topology that allows us to distinguish between different types of users. In order to do so, we follow two different approaches.

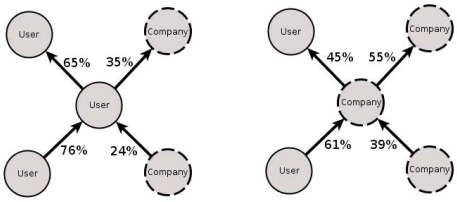
Our first approach consist on finding a set of attributes of the nodes of the network, such that nodes in the same category have similar values while nodes in different categories present significantly different values. However, instead of taking into account local semantic node attributes that are usually present in OSN data, we restrict ourselves to information that can be extracted from the network itself, that is, information that can be obtained from the mere existence of nodes and the relationships created between them. There exist many different network metrics describing the importance of a node in the network that could be used as node attributes in a classification process. However, calculating some of these measures is computationally expensive for large networks and, as we will see, it may not be necessary to use them. In this section, we describe two network properties that are easy to compute and from which we build our classifiers: node degree and clustering coefficient.

One of the most direct metric for nodes is their degree. In Twitter, node degree alone already gives quite a lot of information about the type of user of the network. It is immediate to observe that, for instance, celebrities have really high indegree due to the fact that lots of fans follow what the celebrity says on Twitter, while not so popular users tend to have smaller indegree values.

Figure 1(a) shows an indegree versus outdegree scatter plot for nodes classified in two different categories, users and companies, corresponding to the binary classification problem. Each of the samples is represented by a single mark, with companies represented by crosses and users by dots. We can appreciate that most of the individual users have an indegree similar to its outdegree. On the contrary, companies tend to have notable differences between indegree



(a) Scatter plot of node indegree and outdegree for users and companies (dataset 1)



(b) Relationship profile for users and companies (dataset 1)

Fig. 1. Information for the binary classification problem

and outdegree, showing higher indegrees than outdegrees on most of the cases. However, we can also observe that there are some companies that present an indegree versus outdegree ratio similar to that showed by individual users. This indicates us that although some kind of classification can be done with node degrees, it will not lead us to specially good results.

Clustering coefficient is a measure of how well connected the neighborhood of a node is. When the neighborhood of a node is fully connected, the clustering coefficient is 1, whereas when there are no connections between one node's neighbors, the clustering coefficient is 0. For the Twitter network, this means that when a user's followers also create following relationships between them, the clustering coefficient of the user is high; in contrast, when user's followers are not connected between them, clustering coefficient becomes low. Specifically, clustering coefficient of a node v is defined as the number of connections between v's neighbors divided by the number of possible connections that could exist between them. Social networks are known to exhibit high clustering coefficients but, do different kinds of users present different cluster coefficient values? Our intuition says that normal users will present higher clustering coefficient values than elite users because of their use of the network: elite users such as celebrities or media make a usage of the network similar to that of a traditional diffusion media and, therefore, they will not exhibit this high clustering found in social networks.

The second approach that we use to classify Twitter users is by taking advantage of the type of users to which they are connected. Recent studies [2] show that Twitter users tend to listen to other users in the same category. Then, we can use the categories assigned to a node's communication neighbors in order to determine the category of that node. We want to use a similar approach to differentiate users from the network topology, taking advantage of which types of user are they following and which types of users are following them up.

In order to do so, we analyze if there are any differences between categories in terms of network linkage. We compare how users of each of the defined categories

are connected to users in any category. Figure 1(b) shows the relationship profile for the binary classification problem (individual users versus companies). Values on the edges represent the percentage of existing relationships of the specific type from the total of the outgoing or incoming relationships. Therefore, users outgoing relationships are directed, in 65% of cases, to other users, and in 35%, to companies. On the contrary, companies outgoing relationships are directed to users or other companies in a more equitable way (45% for users, 55% for companies). Incoming edges also present the same kind of phenomenon, with users being followed mostly by users and companies having a more equitable incoming profile. All together, these differences on the relationship profiles of the two different categories will be exploited to build our classifier.

5 Classifier Proposal

In this section, we propose a relational classifier that takes advantage of the homophily showed by OSN users. However, in order to classify a user depending on the classes of his neighbors, we need to actually know the classes of those neighbors, which at the same time presents this very same problem. Therefore, we can not use the relationship profile of a user to classify that user at an initial stage. We need to assign a prior classification to nodes before using the relationship profile to classify users. So we design our Twitter user classifier as a two module classifier: an initial classifier, which makes a first labeling of users into the desired categories; and a relational classifier, which uses the results on the previous classifier to exploit the relationship profile to classify. The performance of the initial classifier is not critical since its results are only used as inputs for the relational classifier. Moreover, the relational classifier can be applied iteratively, so that the results of one execution of the relational classifier can be used as new labels for a new execution of the relational classifier.

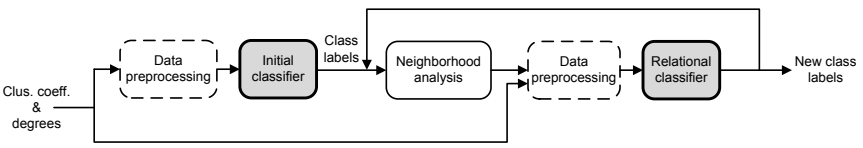


Fig. 2. Classifier modules scheme

The architecture of this classifier is shown in Figure 2. It is possible to appreciate that, apart from the two classifier modules, there is also a data preprocessing module applied just before the classifiers and a neighborhood analysis stage that analyses the current labels and generates the relationship profiles of each of the samples. Figure 2 also indicates that the class labels leaving the relational classifier are used in the successive iterations as input labels for the neighborhood analysis module. It is worth mention that classification at stage $t + 1$ uses only labels designed at stage t.

This architecture is similar, to some extent, to the one used by NetKit [5]. In such proposal, the authors define the three main components of a node-centric network learning system: the non-relational model, the relational model, and the collective inference method. The non-relational model uses only local information of the entities that are going to be classified. This non-relational model can be used to create priors which are used as the initial state for the relational model. The relational model uses the relations of the entities in the network and the attributes of the neighbors of those entities. Finally, the collective inference method faces the problem of having to classify a user depending on his neighbors classification, which applies recursively to all connected nodes of the dataset. Although our architecture resembles the one used by NetKit, note that there are many differences between both approaches: the non-relational model of our work does not use node attributes but properties that can be extracted from the network structure itself; no attributes are neither used on the relational phase on our proposal; and, as we will see, we instantiate the relational model with a Support Vector Machine classifier, which differs from all the relational models proposed in [5]. Note that by not using any node attributes in our classification, our proposal can be used with naively anonymized graphs whose node attributes are removed before being released.

Classification problems consist on assigning samples of an input set into a given number of categories. In our scenario, we denote our category set as $\mathfrak{C} = \{\mathfrak{c}_k, \text{ for } k = 1, \cdots, m\}$. Then, our classifier will assign a value in $\mathfrak{C}$ to each node v_i from the input dataset. However, since our classifier uses an iterative procedure to estimate the correct category of each node, we denote by $cat^t(v_i)$ the predicted category in the t iteration. Then, $cat^0(v_i)$ is the predicted category for node v_i performed by the initial classifier.

5.1 Initial Classifier

The initial classifier uses the structural node properties described in Section 4: node degree and clustering coefficient. So for this classifier, each node is represented as a 3 dimensional sample, which describes its clustering coefficient, indegree, and outdegree values. Note that although these attributes are structural properties of the nodes in the network, there is no need to known the attributes of the neighbors of a given node to compute them.

The initial classifier is built upon a Support Vector Machine classifier[3] with soft margins, a Gaussian Radial Basis Function kernel, and a scaling factor of 1. Our dataset contains some weird nodes that, although being labeled as members of one type, they exhibit values on the computed attributes very similar to those shown on nodes of the other types. For this reason, we use a soft margin classifier in order to find a solution that better distinguishes the majority of the nodes while neglecting to classify these outliers. Using a Radial Basis Function as kernel shows good enough results for the initial classifier. Other kernels such as high

[3] For those readers not familiar with Support Vector Machines, references [22] and [23] cover the basics of such techniques.

degree polynomials also offer similar performance results but they need more computational time to be able to train the classifier.

After all this data processing is done, the support vectors are computed from the test dataset with Quadratic programming method and the final result is an initial classification, where each node v_i has been assigned to a category $\mathfrak{c}_k$ for some $k \in \{1, \cdots, |\mathfrak{C}|\}$.

5.2 Relational Classifier

Once we have an initial classification of users into categories, obtaining $cat^0(v_i) = \mathfrak{c}_k \ \forall i = 1, \cdots, n$, we can use this information to build a second classifier which takes into account to what types of users is each user connected to. We assume that the class of a node depends only on the classes of their direct neighbors, such that the probability of a node belonging to a given class is independent of the rest of the graph but his immediate neighborhood. This makes the problem of inferring class membership more tractable. Then, in a similar way than with the Class-Distribution Relational neighbor Classifier [5,24,25], we construct the node v_i class vector $CV(v_i)$ as the vector of summed linkage weights to the various known classes. Since our topological social graph does not have weights on the edges, we assume that each edge has a weight of exactly 1. So the kth position of the class vector $CV^{(t)}(v_i)_k$ computed in the t iteration of the process is the number of neighbors of v_i within the predicted class in the previous classification stage $t-1$. However, since we are dealing with directed graphs, we extend the class vector to contain two different values for each category, corresponding to the predecessors and the successors of the analyzed node. In this way, each $CV(v_i)$ vector component has exactly 2 dimensions, the first corresponding to the successors and the second corresponding to the predecessors:

$$CV^{(t)}(v_i)_{k,1} = |\{v_j \in \Gamma(v_i) \text{ s.t. } cat^{t-1}(v_j) = \mathfrak{c}_k\}|$$

$$CV^{(t)}(v_i)_{k,2} = |\{v_j \in \Gamma^{-1}(v_i) \text{ s.t. } cat^{t-1}(v_j) = \mathfrak{c}_k\}|$$

Following the scheme showed in Figure 2, $CV^{(t)}(v_i)$ is the result of the neighborhood analysis box in the t-th iteration, which is used as input for the relational classifier, after being properly preprocessed. As depicted in Table 2, apart from these attributes derived from the relationship profile of the users, we also add the 3 dimensions used in the initial classifier to each of the samples. In this manner, we use as much information as we have at each stage to conduct our classification.

Once the vectors for each of the samples have been constructed, we build a classifier with the same parameters than the initial classifier: using Support Vector Machines with a Gaussian Radial Kernel Function and a scaling factor equal to 1.

This relational classifier is applied iteratively. Since the output of the refinement classifier should be better than that of the initial classifier, we can use the output of the relational classifier to compute new values for the percentages of

users in outgoing and incoming edges, and then apply the relational classifier again to improve classification performance. Ideally, we would like to run the relational classifier iteratively as many times as needed until the results converge. However, this method may not always converge, so some other termination condition has to be set to stop the iterative process. In our case, we fixed a maximum amount of iterations and considered as final results those obtained when that maximum amount of iterations is reached.

6 Experimental Results

As we have already mentioned, the adversary goal is to classify Twitter users using a naively anonymized copy of the social graph. Such anonymized copy does not include node labels but it has exactly the same structure than the original graph: no edges nor nodes have been added nor deleted. However, in our scenario, we go one step beyond naive anonymization and we also remove all node attributes, so that the adversary cannot use any semantic information in the classification process.

In order to perform the attack, the adversary is able to learn labels for some subset of the nodes of the graph a priori. These nodes for which the attacker is able to obtain labels beforehand are used as training samples in the classification process. Then, the adversary infers labels for the rest of the dataset, learning information from users for which no prior information could be found.

The data to perform the experiments for the validation of the proposed attack have been obtained from the Twitter network. More precisely, we have collected two different Twitter samples using a Breadth First Search scheduler algorithm, each of one starting from a different user in the Twitter network. To obtain each of the corresponding social graphs, we have explored around 300 nodes, which lead us to discover almost a million (936.423) different Twitter users. We use those graphs to test our classifier architecture with two different classification objectives: a binary classification attack (individual users versus companies) and a multiclass classification attack with five different categories (four elite users categories plus a non-elite user class).

We evaluate the success of the attack using the classical machine learning approach of repeated random sub-sampling validation. We use the same approach than in [5] to create training and test sets. Given a crawled graph $G = (V, E)$, we randomly pick a subset of labeled nodes $V_{train} \subset V$ to be used as the training set. Then, the test set V_{test} is defined as the rest of the nodes, so $V_{test} = V \setminus V_{train}$. Therefore, we are facing a within-network classification problem, having a scenario with labeled nodes linked with nodes for which the class in unknown.

The experimentation methodology is the same for both classification attacks and for both datasets. We repeat the process of randomly selecting test and training sets, building classifiers based on the training information and evaluating the results with the test set 100 times. Then, we use mean correct classification rate values to evaluate the attack success. By repeating the experiment 100 times we pursue to minimize the artifacts of selecting specific samples for the training and test sets.

We repeat each of the experiments for different training set sizes (65%, 50%, 35%, and 20%) in order to evaluate the effects of the number of labeled nodes in the attack success.

The next subsections review the specific configuration parameters used in each of the two different classification attacks, the class labeling ground of truth definition for each one, and the obtained results when using our classifier architecture to classify samples of the collected datasets.

6.1 Binary Classification Attack

Firstly, we use the classifier described in the previous section to classify users in two different categories: individual users and companies (or organizations). While some of the collected Twitter accounts are used by individual users in a personal capacity, others are used by companies or organizations to promote themselves, their products, or to maintain a link with their costumers or members. Our goal with this attack is to distinguish between these two different uses of Twitter. Following the introduced notation, our binary classification attack consists on classifying users into two different categories ($|\mathfrak{C}| = 2$), where $\mathfrak{c}_1$ = "individual" and $\mathfrak{c}_2$ = "company".

In order to train the classifier and to be able to evaluate the success of the attack, we perform a manual labeling of all users in the two collected graphs, marking each user as either an individual user or a company. Table 1 shows the number of users belonging to each of the categories. We can observe that there is a similar distribution of samples of each of the two classes in both datasets.

Table 1. Number of users in each category

Category	Dataset 1	Dataset 2
$\mathfrak{c}_1$ = "individual"	159	161
$\mathfrak{c}_2$ = "company"	144	174

At the beginning of the classification process, the initial classifier receives as input data the 3-dimensional samples that contain the nodes' clustering coefficient, indegree and outdegree (upper vector in Table 2). Its output is a set of binary class labels that classify all nodes v_i within a given category $\mathfrak{c}_k$, obtaining $cat^0(v_i) = \mathfrak{c}_k$. Such information is used during the neighborhood analysis to construct the 7 dimensional samples used by the relational classifier. These 7 dimensions (lower vector in Table 2) comprehend the same 3 dimensions included in the initial classifier plus 4 new dimensions created by analyzing to which classes does every node connect; in this case, the number of successors classified as individual users, $CV^{(t)}(v_i)_{1,1}$, the number of successors classified as companies, $CV^{(t)}(v_i)_{2,1}$, the number of predecessors classified as individual users, $CV^{(t)}(v_i)_{1,2}$, and the number of predecessors classified as companies, $CV^{(t)}(v_i)_{2,2}$.

Table 2. Sample dimensions for the binary classification problem

cc	Indeg	Outdeg
Initial		

cc	Indeg	Outdeg	CV_1 = individual		CV_2 = company	
Initial			In	Out	In	Out

Figure 3 draws the results of the attack success, showing the rates of correct node classification at each iteration step (x axes) for both datasets (solid lines for dataset 1, dashed lines for dataset 2) for different training and test set sizes (circles for 65% of samples into the training set, asterisks for 50%, diamonds for 35%, and triangles for 20%). We can observe that, although the initial classification (iteration 0) never gets over a 63% of correctly classified samples, the correct rate increases considerably with the first relational classification stage (iteration 1), and keeps increasing with the following iterations until it stabilizes (around iteration 5), presenting correct rates close to 73% when the training set contains 65% of the nodes of the graph. As it is expected, the attack success increases as the training set size also increases because nodes in the training set are used to construct the relationship profiles of nodes in the test set, so increasing the training set size also increases the amount of correct information available when classifying.

We can also observe that similar results are obtained with both datasets, with differences in the final classification performance being lower than 2%. This seems to indicate that our proposed classifier architecture can be used to attack the Twitter network and classify users in these two classes regardless of the specific part of the network analyzed.

6.2 Multiclass Classification Attack

Our second experiment uses the same classifier architecture to group users in five different categories (the four used by [2] to describe elite users plus a new category to describe non-elite users). So this second attack intends to classify Twitter users in one of the following five categories ($|\mathfrak{C}| = 5$): $\mathfrak{c}_1$ = "normal user", $\mathfrak{c}_2$ = "blogger", $\mathfrak{c}_3$ = "celebrity", $\mathfrak{c}_4$ = "media", and $\mathfrak{c}_5$ = "organization".

In order to build the classifier for this multiclass problem, we use a combination of binary Support Vector Machine classifiers with one-versus all methodology: we construct 5 binary classifiers, each of them considering positive samples the nodes of one class and negative samples the nodes of all other classes. Then, we assign each test sample to the class that classifies it with the greatest margin. Individual binary classifiers are build with the configuration detailed in Section 5.1.

In this scenario, in order to train the classifier and to evaluate the success of the attack, we perform an automatic user labeling process using the Twitter lists feature (following a similar procedure than in [2]). In this case, we use just

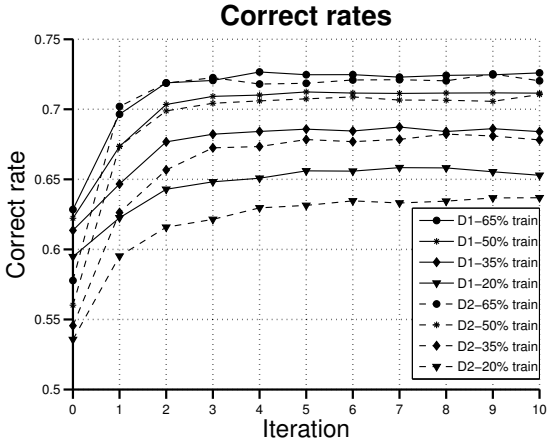


Fig. 3. Binary classification attack success

Table 3. Number of users in each category

Category	Users
$\mathfrak{c}_1$ = "normal user"	16(5%)
$\mathfrak{c}_2$ = "blogger"	46(14%)
$\mathfrak{c}_3$ = "celebrity"	86(26%)
$\mathfrak{c}_4$ = "media"	93(27%)
$\mathfrak{c}_5$ = "organization"	94(28%)

one of our datasets, since the list labels were defined with English words and the other dataset contains mostly non-English speaking users.

Unlike the binary classification attack, the number of nodes assigned to each of the categories is distributed unequally in the multiclass classification problem, with some categories containing very few nodes and others with as much as 28% of all nodes. Table 3 shows the number of users labeled in each category.

The inputs for the initial classifier are the same 3-dimensional samples than in the binary case (clustering coefficient, indegree, and outdegree). However, the inputs of the relational classifier have now 13 dimensions: the initial 3 dimensions plus 10 relational dimensions obtained by studying the 5 classes in both outgoing $(CV^{(t)}(v_i)_{j,1}$ for $j = 1, \cdots, 5)$ and incoming edges $(CV^{(t)}(v_i)_{j,2}$ for $j = 1, \cdots, 5)$. Table 4 shows each of the dimensions for the initial and relational classifiers.

Figure 4 shows the results of the multiclass attack for different training and test set sizes. We can observe that, although the initial classification (iteration 0) is always below 50%, the correct rate increases considerably with the first relational classification stage (iteration 1). However, further iterations of the relational classification do not substantially increase the correct rate value. As with the binary classification problem, the performance of the classifier decreases when the training set size also decreases.

Table 4. Sample dimensions for the multiclass classification problem

cc	Indeg	Outdeg
Initial		

cc	Indeg	Outdeg	CV_1 =user		CV_2 =blog		CV_3 =celeb		CV_4 =media		CV_5 =org	
Initial			In	Out	In	Out	In	Out	In	Out	In	Out

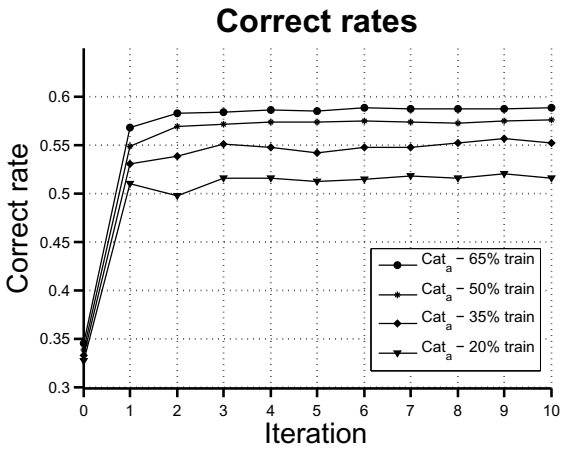


Fig. 4. Multiclass classification attack success

7 Conclusions and Further Work

In this paper, we have shown that it is possible to classify OSN users using a naively anonymized copy of a social graph. By using the proposed architecture we analyze two different classification attacks, aimed to show that classification is feasible at different levels. Our goal was also to demonstrate that the information found in the social graph was enough to perform classification. In the binary classification attack, our architecture is able to achieve success rates up to 65% assuming that the attacker knows only the a priori classification of the 20% of users. In the multiclass classification attack the success rate is lower, up to 52% when the attacker knows only the 20% of correct labels. Furthermore, by using data from two almost disjoint user set samples (there is just one user from the 936.423 collected which appears in both datasets), we show that social graph structure is indeed enough to classify users and that this is not a local phenomenon appearing just in a specific isolated set of users of the network.

Given the faced scenario, two different approaches are the natural continuation of this work. On one hand, we can assume that the attacker is able to obtain some attributes of the nodes, being able to improve the classification using this information. Further work remains to be done with this approach to integrate both information from the social structure and semantic information found in

node attributes to improve OSN user classification. On the other hand, we can also assume that the attacker obtains a copy of the graph released after an anonymization process other that the naive anonymization. In this case, further work also remains to be done to analyze the impact of different anonymization techniques on the classification performance.

References

1. Boyd, D., Ellison, N.B.: Social network sites: Definition, history, and scholarship. Journal of Computer-Mediated Communication 13(1) (2007)
2. Wu, S., Hofman, J.M., Mason, W.A., Watts, D.J.: Who says what to whom on twitter. In: Proc. of World Wide Web Conference, WWW 2011 (2011)
3. Jernigan, C., Mistree, B.F.T.: Gaydar: Facebook friendships expose sexual orientation. First Monday 14(10) (2009)
4. Westin, A.: Privacy and Freedom. Atheneum (1970)
5. Macskassy, S.A., Provost, F.: Classification in networked data: A toolkit and a univariate case study. J. Mach. Learn. Res. 8, 935–983 (2007)
6. Macskassy, S.A., Provost, F.: A simple relational classifier. In: Proc. of the 2nd Workshop on Multi-Relational Data Mining, KDD 2003, pp. 64–76 (2003)
7. Chakrabarti, S., Dom, B., Indyk, P.: Enhanced hypertext categorization using hyperlinks. In: SIGMOD 1998: Proc. of the 1998 ACM SIGMOD International Conference on Management of Data, vol. 27, pp. 307–318. ACM Press, New York (1998)
8. Lu, Q., Getoor, L.: Link-based classification using labeled and unlabeled data. In: Proc. of the ICML 2003 Workshop on the Continuum from Labeled to Unlabeled Data, Washington, DC (2003)
9. Jensen, D., Neville, J., Gallagher, B.: Why collective inference improves relational classification. In: KDD 2004: Proc. of the 2004 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 593–598. ACM Press, New York (2004)
10. Geman, S., Geman, D.: Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-6(6), 721–741 (1984)
11. Neville, J., Jensen, D.: Iterative classification in relational data. In: AAAI 2000 Workshop on Learning Statistical Models from Relational Data (2000)
12. Gallagher, B., Eliassi-Rad, T.: An examination of experimental methodology for classifiers of relational data. In: Proc. of the 7th IEEE Int. Conf. on Data Mining Workshops, ICDMW 2007, pp. 411–416. IEEE Computer Society (2007)
13. Carvalho, V.R., Cohen, W.W.: On the collective classification of email "speech acts". In: SIGIR 2005: Proc. of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 345–352. ACM, New York (2005)
14. Bhagat, S., Cormode, G., Rozenbaum, I.: Applying Link-Based Classification to Label Blogs. In: Zhang, H., Spiliopoulou, M., Mobasher, B., Giles, C.L., McCallum, A., Nasraoui, O., Srivastava, J., Yen, J. (eds.) WebKDD/SNA-KDD 2007. LNCS, vol. 5439, pp. 97–117. Springer, Heidelberg (2009)
15. Hay, M., Miklau, G., Jensen, D., Weis, P., Srivastava, S.: Anonymizing Social Networks. Technical report (2007)

16. Zheleva, E., Getoor, L.: Preserving the Privacy of Sensitive Relationships in Graph Data. In: Bonchi, F., Ferrari, E., Malin, B., Saygin, Y. (eds.) PinKDD 2007. LNCS, vol. 4890, pp. 153–171. Springer, Heidelberg (2008)
17. Liu, K., Terzi, E.: Towards identity anonymization on graphs. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 93–106. ACM, New York (2008)
18. Zhou, B., Pei, J.: Preserving privacy in social networks against neighborhood attacks. In: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, ICDE 2008, pp. 506–515. IEEE Computer Society, Washington, DC (2008)
19. Zou, L., Chen, L., Özsu, M.T.: k-automorphism: a general framework for privacy preserving network publication. Proc. VLDB Endow. 2(1), 946–957 (2009)
20. Ford, R., Truta, T.M., Campan, A.: P-sensitive k-anonymity for social networks. In: Stahlbock, R., Crone, S.F., Lessmann, S. (eds.) DMIN, pp. 403–409. CSREA Press (2009)
21. Knuth, D.E.: Art of Computer Programming: Fundamental Algorithms, 3rd edn., vol. 1. Addison-Wesley Professional (July 1997)
22. Hearst, M., Dumais, S., Osman, E., Platt, J., Scholkopf, B.: Support vector machines. IEEE Intelligent Systems and their Applications 13(4), 18–28 (1998)
23. Manning, C.D., Raghavan, P., Schtze, H.: Support vector machines & machine learning on documents. In: Introduction to Information Retrieval, pp. 319–348. Cambridge University Press (2008)
24. Perlich, C., Provost, F.: Distribution-based aggregation for relational learning with identifier attributes. Machine Learning 62(1-2), 65–105 (2006)
25. Rocchio, J.: Relevance Feedback in Information Retrieval, pp. 313–323. Prentice Hall (1971)

A Formal Derivation of Composite Trust

Tim Muller and Patrick Schweitzer

University of Luxembourg, CSC and SnT
{tim.muller,patrick.schweitzer}@uni.lu

Abstract. Trust appears in asymmetric interactions, where one party (the active party) can easily betray a stakeholder (the passive party). Over the Internet, the amount of information that a passive party can use to determine the integrity of an active party is often limited. The scenario where there is only one passive party and one active party is well studied, and has been solved under some assumptions. We generalize the setting to allow for more parties. In particular, the paper contains a formal derivation of conjunction (and disjunction) of trust opinions.

1 Introduction

Trust has a diverse meaning to different people. Consequently, definitions of trust in the literature vary. A definition of trust can be found in [1]: "First, one trusts another only relatively to a goal, i.e. for something s/he wants to achieve, that s/he desires. (..) Second, trust itself consists of beliefs. Trust is a mental state (..) about the behavior (..) relevant for the result." From that definition, we see that the authors clearly see trust as a mental state, regarding interactions where the result (rather than the intention) is important. In [2] an economical perspective is taken: "Trust is a psychological state comprising the intention to accept vulnerability based upon positive expectations of the intentions or behavior of another." Here, it is clear that intention is relevant, and trust is still about a mental state and a specific interaction. Existing reputation systems must take trustworthiness as an objective property, rather than a mental state, in order for reputation to have meaning. Definitions with a psychological accent often do not depend on interactions, but tie trust to agents, as written in [3]: "Trust in things or people entails the willingness to submit to the risk that they may fail us, with the expectation that they will not (..)." These definitions, as well as informal intuitions share properties that are difficult to characterize. In this paper, we take the view that trust helps us to reason about trust systems (such as reputation systems and recommender systems) on the Internet. This means that trustworthiness, or *integrity*, is taken as an inherent property of the agents. It also means that agents trust each other with respect to interactions within the system. Furthermore, it means that only the result of the interaction matters, not the intention of the agent.

In each of the previously discussed definitions of trust an agent makes an assessment of another agent's future behavior using information they have gathered in the past. Such an assessment is called a *simple trust opinion*. If an agent

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 132–148, 2013.

makes an assessment about the future behavior of several agents, it is called a *composite trust opinion*. A trust opinion does not only predict the future behavior that is most likely, but also indicates the certainty of the prediction. In this paper, we formalize the aforementioned notion of trust, using trust opinions. To obtain meaningful results, we must specialize the notion of trust.

In an interaction, there are several parties that have an agreement. There is at least one *active party*, who has an opportunity to ignore the agreement, and there is one *passive party*, which cannot affect the outcome of the interaction and may be harmed if active parties ignore the agreement. If an active party adheres to the agreement, we say the active party's behavior is *good behavior*, if he fails to adhere, we say it is *bad behavior*. Since the passive party may be harmed if one of the active parties shows bad behavior, the passive party may form a trust opinion about the active parties before (potentially) interacting. If an agent forms a (composite) trust opinion about (several) agents, he is called the *subject*. The combination of the active parties concerning the potential interaction is called the *target*. To express composite trust opinions we denote the target in propositional logic, where atomic propositions represent good or bad behavior of active parties. To illustrate the use of composite trust, consider the following example.

Example 1. Take an imaginary web service, CLOUD, that offers computational power to users, by CPU-scavenging in a similar fashion to BOINC [4], i.e. CLOUD is a grid. A user that delegates a computation is a client, and a user that offers CPU cycles is a provider. Unlike BOINC, the CLOUD system is a commercial system, where clients pay for computations, and providers get paid for offering computational power.

The identity of the machines in CLOUD is visible, and users can delegate computations to specific (groups of) machines. The infrastructure of CLOUD is very open, which means that malicious users can easily join as a provider. Malicious users may sometimes take shortcuts in the computation, providing wrong results. Furthermore, non-malicious users may prematurely terminate a computation before a result is provided, for example, when the computer shuts down, restarts or drops the network. It may occur that a single computation is delegated to a group of computers working concurrently to reduce latency. It may also occur that a single computation is delegated to more than one (group of) provider(s), to avoid extra latency when one of the (groups of) providers fails.

A client, A, on CLOUD has an instance of an NP-complete problem, and sends the problem to a provider, D, and a copy of the problem to a pair of concurrent providers, B and C. See Figure 1 for a visual representation of the interaction.

In our terminology, clients are passive parties (i.e. potential subjects), and providers are active parties (i.e. potential targets). Good behavior for a provider is delivering a correct result within a specified time frame. Bad behavior for a provider is returning a wrong result, returning it too late, or not at all. Since a client can quickly verify a (positive) solution to an NP-complete problem,

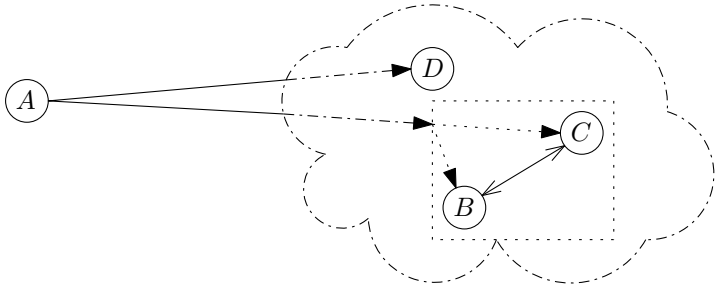


Fig. 1. The two outgoing arrows from A are delegations for a computation. One for D, the other is split and runs concurrently on B and C.

correct and incorrect solutions can easily be distinguished. Hence, it suffices for the subject A to receive at least one correct result within the specified time frame from the target. If either the single provider D or both other providers B and C provide the correct result in time, the whole target's behavior is considered good. We can denote this composite trust opinion as $D \vee (B \wedge C)$.

The subject not only wants to know the probability that the target succeeds, but also the uncertainty. If the probabilities b, c and d of B, C and D succeeding are independent, then one may anticipate that the expected probability of the target succeeding to be $d + b \cdot c - d \cdot b \cdot c$. We formally show the foresight on this trust opinion to be correct in Section 4.

To derive trust opinions, as the ones in the previous example, in a formal way, we need to define the context. We assume to be in an environment where all information comes from interactions between passive and active parties and that active parties operate independently of each other. Furthermore, we assume that each active party has an (hidden) integrity parameter that represents the probability of good behavior in unknown contexts. Therefore, in our framework, a trust opinion is a probability distribution over integrity parameters. From such a probability distribution, one can derive the expected probability of good behavior and the uncertainty of the estimate. In Section 3, we define the context (including trust opinions) formally.

The framework that we operate in is inspired by work in [5] and [6], where the authors independently derived a formal trust model (the *beta model*) of trust opinions. The context of the beta model is very similar to ours, but only allows simple trust opinions, not composite trust opinions. In the beta model, probability distributions known as beta distributions [7] represent trust opinions.

The beta model inspired several very popular extensions, such as Subjective Logic [8], TRAVOS [9] and CertainTrust [10]. Like our trust model, Subjective Logic contains conjunction and disjunction operators, and CertainTrust has been extended to CertainLogic [11] which also contains these operators. Unlike in our trust model, their trust model composite trust opinions are beta distributions. In Section 4, we show that models where composite trust opinions are beta distributions must violate reasonable assumptions

Not all models with conjunction, disjunction and uncertainty are based on the beta model; important examples are Fuzzy logic [12] and Dempster-Shafer theory [13,14]. Trying to apply our results to these models is more difficult, due to some inherently different assumptions and approaches.

There are also trust models that are based on the beta model, which have been extended beyond it. Often, they are the result of tweaking the assumptions of the beta model. In [15], for example, the assertion that behavior is good or bad is generalized into an assertion that a behavior value is selected from a range of values. For CLOUD, this means that incorrect results are distinguished from late results, or lack of results. In [16], the assumption of equal weight to all interactions is dropped. In the case of CLOUD, this implies that interactions that lie further in the past are less relevant than more recent interactions; or that providers behave differently during peak-load periods and off-peak periods.

In Section 2 of this paper, we introduce the necessary concepts from probability theorem. On the basis of these concepts, we formalize the assumptions necessary to reason about trust in our context, in Section 3. We argue that it is useful to have such a collection of simple assumptions from which to derive trust opinions (as opposed to defining how to derive trust opinions directly). That trust operators can be derived from the assumptions is shown in Section 4, where we derive conjunction and disjunction.

2 Preliminaries

The setting of the model is probabilistic in nature. We require the following concepts from probability theory (see e.g. [17,18]).

Definition 1 (σ-algebra, measure, probability measure). *Let Ω be a set of events. A set $\mathcal{F}$ of subsets of Ω is called a σ-algebra if the following three properties hold.*

1. $\emptyset \in \mathcal{F}$.
2. *If* $A \in \mathcal{F}$ *it follows that* $\Omega \setminus A \in \mathcal{F}$.
3. *If* $A_1, A_2, \ldots \subset \mathcal{F}$ *it follows that* $\bigcup_n A_n \in \mathcal{F}$.

Let P be a map from $\mathcal{F} \to \mathbb{R} \cup \{\infty\}$. Then, this map is called a measure if

1. $P(\emptyset) = 0$.
2. $P(A) \geq 0$ *for all* $A \in \mathcal{F}$.
3. *If* $A_1, A_2, \ldots \subset \mathcal{F}$ *such that* $A_k \cap A_l = \emptyset$ *for all* $k \neq l$, *it follows that* $P(\bigcup_n A_n) = \sum_n P(A_n)$.

If P maps to $[0, 1]$ and $P(\Omega) = 1$, it is called a probability measure.

The tuple $(\Omega, \mathcal{F})$ from Definition 1 is called a measurable space. The triple $(\Omega, \mathcal{F}, P)$ is called a measure space. If P is additionally a probability measure, the triple is called a probability space.

Definition 2 (Random Variable). *Let $(\Omega, \mathcal{F}, P)$ be a probability space and $(E, \mathcal{E})$ a measurable space. A mapping $X\colon \Omega \to E$ is a random variable, if*

$$\{\omega \in \Omega | X(\omega) \in B\} \in \mathcal{F} \text{ for all } B \in \mathcal{E}.$$

When Ω and E are at most countable, the σ-algebras $\mathcal{F}$ and $\mathcal{E}$ can be assumed to be the power sets over Ω and E, respectively.

In probability theory, the expression $\{\omega \in \Omega | X(\omega) \in B\}$ is often abbreviated to $\{X \in B\}$.

Definition 3 (Probability space of a random variable). *Let $(\Omega, \mathcal{F}, P)$ be a probability space, $(E, \mathcal{E})$ a measurable space and $X\colon \Omega \to E$ a random variable. Then $P_X(B) := P(\{X \in B\}), B \in \mathcal{E}$ defines a probability measure P_X on $(E, \mathcal{E})$.*

The expression $P(\{X \in B\})$ is usually shorthanded to $P(X \in B)$.

Definition 4 (Distribution of a random variable). *The probability measure P_X is called the distribution of the random variable X.*

The probability space $(E, \mathcal{E}, P_X)$ is called discrete, if E is at most countable.

Definition 5 (Independence of random variables). *Let $(\Omega, \mathcal{F}, P)$ be a probability space and let $X_1, \ldots, X_n$ be n random variables (over Ω) with values in the measurable spaces $(E_i, \mathcal{E}_i)$, $i \in \{1, \ldots, n\}$. The random variables $X_1, \ldots, X_n$ are called independent, if for arbitrary $B_1 \in \mathcal{E}_1, \ldots, B_n \in \mathcal{E}_n$, the events $\{X_1 \in B_1\}, \ldots, \{X_n \in B_n\}$ are independent.*

This definition is equivalent to the following.

$$X_1, \ldots, X_n \text{ indep. } \Leftrightarrow P(X_1 \in B_1, \ldots, X_n \in B_n) = \prod_{i=1}^{n} P_{X_i}(B_i) \text{ for } B_i \in \mathcal{E}_i.$$

As shorthand notation we write $X \perp\!\!\!\perp Y, Z$ when X, Y, Z are independent.

Definition 6 (Conditional independence of variables). *Let $(\Omega, \mathcal{F}, P)$ be a probability space and let X, Y, Z be random variables (from Ω) with values in the measurable spaces $(E_i, \mathcal{E}_i)$, $i \in \{X, Y, Z\}$. Two random variables X and Y are conditionally independent given the variable Z if*

$$P(X \in A, Y \in B | Z \in C) = P(X \in A | Z \in C) P(Y \in B | Z \in C).$$

for each $A \in \mathcal{E}_X, B \in \mathcal{E}_Y$ and $C \in \mathcal{E}_Z$.

As shorthand we write $P(X, Y|Z) = P(X|Z)P(Y|Z)$, $(X \perp\!\!\!\perp Y)|Z$ or even $X \perp\!\!\!\perp Y|Z$. Note that the definition is equivalent to $P(X|Y, Z) = P(X|Z)$.

Theorem 1 (Law of total probability). *Let $(\Omega, \mathcal{F}, P)$ be a probability space, A and C events and let $B_1, \ldots, B_n$ be a partition in that probability space. Then*

$$P(A|C) = \sum_{i=1}^{n} P(A|B_i, C) P(B_i|C).$$

The law of total probability also holds for continuous random variables X, and Y with positive density functions f_X and f_Y, respectively.

$$f_Y(y) = \int_{-\infty}^{\infty} f_Y(y|X = x) \cdot f_X(x)\,\mathrm{d}x.$$

Theorem 2 (Bayes' law for conditional probabilities). *Let $(\Omega, \mathcal{F}, P)$ be a probability space and B and C events and let $A_1, \ldots, A_n$ be a partition in that probability space. Then*

$$P(A_j|B, C) = \frac{P(B|A_j, C)P(A_j|C)}{P(B|C)} = \frac{P(B|A_j, C)P(A_j|C)}{\sum_{i=1}^{n} P(B|A_i, C)P(A_i|C)}.$$

Note that in this form Bayes' theorem also holds for variables (instead of events). This is true for discrete random variables, continuous random variables as well as a mixture of discrete and continuous random variables. If continuous variables are involved, they need to have a positive density function.

Theorem 3 (Product distribution). *Let X and Y be two independent continuous variables, with positive probability density functions $f(x)$ and $g(x)$. Then $U = X \cdot Y$ is a continuous random variable with probability density function h. Explicitly*

$$h(u) = \int_{-\infty}^{\infty} \frac{1}{|y|} \cdot f(\frac{u}{y}) \cdot g(y)\,\mathrm{d}y.$$

An important distribution we refer to in the next sections is the beta distribution.

Definition 7 (Beta distribution). *A beta distribution is a family of continuous probability distributions in the interval $[0, 1]$, parameterized by two positive parameters, $\alpha, \beta \geq 1$. The probability density function of a beta distribution with parameters α and β is*

$$f_{\mathrm{B}}(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 y^{\alpha-1}(1-y)^{\beta-1}\,\mathrm{d}y}.$$

The expression under the fractions is known as the beta function on α and β, and for positive integers α and β, the beta function fulfills $\mathrm{B}(\alpha, \beta) = \frac{(\alpha-1)!(\beta-1)!}{(\alpha+\beta-1)!}$.

To quantify information, we define a notion of entropy as in [19].

Definition 8 (Entropy). *The entropy H of a discrete random variable X with possible values $x_1, ..., x_n$ for $n \in \mathbb{N}$ is given by $H(X) = \mathbf{E}(I(X))$, where $\mathbf{E}$ is the expected value and $I(X)$ is the random variable denoting the information content of X. If p denotes the probability mass function of X and $c \in \mathbb{N}$, then the entropy can explicitly be written as*[1]

$$H(X) = \sum_{i=1}^{n} p(x_i)\, I(x_i) = \sum_{i=1}^{n} p(x_i) \log \frac{1}{p(x_i)}.$$

[1] In our considerations the base of the logarithm is not important.

If $p(x_i)$ *is equal to* 0 *for some* $i \in \{1, \ldots, n\}$ *and* $n \in \mathbb{N}$*, then* $p(x_i)\log(p(x_i)^{-1})$ *is taken to be* 0*.*

Entropy can be extended for continuous random variables X *ranging from* a *to* b*, with probability density function* f_X

$$h(X) = \int_a^b f_X(x)\log(\frac{1}{f_X(x)})\,\mathrm{d}x.$$

3 Model Assumptions

On a flea market, you see the sellers face to face, see whether they are well organized and you can determine whether they are popular. On an online e-commerce system, such detailed information is not available. You choose the seller based on previous interactions with this sellers, often by including information about interactions that others claim to have had with these sellers. Our trust model is much more relevant for e-commerce systems (and other online services, such as CLOUD), than it is for an actual flea market.

In this section, we formalize the assumptions that we have for trust in a system based on asymmetric interactions (like transactions in e-commerce systems), where expectations are clearly defined. First, we informally introduce our assumptions with motivations, then we formally state the assumptions as relations between random variables.

To reiterate some assertions from the introduction: Interactions are the building blocks in our trust analysis. Interactions are between a passive party (the subject) and active parties (the target). A subject may form a trust opinion about a target, before the subject interacts as passive party with the active parties in the target. The observed behavior of the active party is objectively classified as good (well) or bad (badly). Furthermore, the probability that the active party behaves well is determined by its integrity parameter p. An agent will most likely exhibit non-probabilistic behavior, and will therefore behave well in some situations and badly in others. However, we do not know the correlation between situations and behaviors, nor do we necessarily know the situation. In the light of this, we can view the integrity p as the chance that an agent is in a situation where his behavior is good (or even where behaving well is in his best interest in some iterative game[2], as in [20]). Lastly, we assume that p neither changes over time nor with respect to the environment. This assumption allows us to treat previous interactions in a mathematically coherent way, since all interactions are equally relevant for the current situation. In the model, an agent will never know the integrity of another agent, but will have an estimate based on these previous interactions.

To formulate the above assumptions in a formal manner, we need to define interactions of agents, integrity parameters of agents, sets of interactions that agents made in the past, and composite targets. To comply to notation used

[2] Agents expect to interact multiple times with other agents, and even if betrayal is profitable on the short run, it may be more profitable to conform on the long run.

in probability theory (Bernoulli, binomial and beta distributions), we refer to good behavior of the active party as success, S, and bad behavior as failure, F. We are often interested in the previous interactions between a passive party and an active party, which we call an *interaction history* of the passive party about the active party. Furthermore, we take an interaction history to be a pair of natural numbers: the first number as representing the number of successes (good behavior by an active party), the second number as representing the number of failures. Let $\mathbf{A}$ denote the set of agents. The targets $\mathbf{T}$ are defined by $\varphi ::= A \in \mathbf{A} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$. For $A, B, C \in \mathbf{A}$, $T \in \mathbf{T}$ and a set of events Ω, we define the following random variables.

- $E_T \colon \Omega \to \{\mathrm{S}, \mathrm{F}\}$ is a discrete random variable modeling the outcome of the corresponding interaction with target T.
- $R_T \colon \Omega \to [0,1]$ is a continuous random variable modeling the (hidden) integrity parameter of target T, defining the probability of success.
- $O_B^A \colon \Omega \to \mathbb{N} \times \mathbb{N}$ is a discrete random variable modeling the interaction history of A about B, representing the past interactions between A as passive party and B as active party.

Recall that a trust opinion is a distribution over the integrity parameter of a target, based on the interaction history about the involved active parties. Hence, if a subject A establishes a trust opinion about a target T, where $B, C, \ldots$ are the active parties in T (denoted $B, C, \ldots \in \mathrm{act}(T)$), the density function looks like $f_{R_T}(x | O_B^A \cap O_C^A \cap \ldots)$. In this setting, the only type of information that is important to the subject, are the interaction histories of this subject. If there are other types of information (interaction histories of others, recommendations, a priori knowledge) available, they can be modeled as additional conditions (on additional random variables).

The definition of the random variables alone does not suffice to compute a query such as $f_{R_{B \wedge C}}(x | O_B^A \cap O_C^A)$. To calculate these trust opinions, we need to provide the dependencies and independencies between the random variables. These (in)dependencies are merely a formal denotation of the assumptions that we have. For a more concise formulation of these (in)dependencies, we introduce sets of random variables.

$$\begin{aligned} \mathbb{E} &:= \{E_T : T \in \mathbf{T}\}, \\ \mathbb{R} &:= \{R_T : T \in \mathbf{T}\}, \\ \mathbb{O} &:= \{O_B^A : A, B \in \mathbf{A}\}, \\ \mathbb{W} &:= \mathbb{E} \cup \mathbb{R} \cup \mathbb{O}. \end{aligned}$$

Let $x \in [0,1]$, $n, k \in \mathbb{N}$ and $\lambda \colon \mathbb{N} \to [0,1]$ be a probability distribution. For all $A, B \in \mathbf{A}$ and $S, T \in \mathbf{T}$ we set up the following dependency and independent relations as our assumptions.

D1 R_A is the uniform distribution on $[0,1]$.
If we know nothing about the integrity of A, we assert all values equally likely. For specific applications, statistical data about behaviors of agents may be used to construct an alternative distribution. A suitable distribution has a probability density function that is non-zero on $(0,1)$.

D2 $P(E_T{=}\mathrm{s}|R_T{=}p) = p$.
We assume that the probability of good behavior is determined by the integrity parameter p.

D3 $E_{S\wedge T} = \mathrm{s}$ iff $E_S = \mathrm{s}$ and $E_T = \mathrm{s}$, for $\mathrm{act}(S) \cap \mathrm{act}(T) = \emptyset$.
We define conjunctions of independent targets in such a way that the conjunction succeeds if both targets succeed.

D4 $E_{S\vee T} = \mathrm{s}$ iff $E_S = \mathrm{s}$ or $E_T = \mathrm{s}$, for $\mathrm{act}(S) \cap \mathrm{act}(T) = \emptyset$.
We define disjunctions of independent targets in such a way that the disjunction succeeds if at least one target succeeds.

D5 There exists a function f, with $R_{S\wedge T} = f(R_S, R_T)$, when $\mathrm{act}(S)\cap\mathrm{act}(T) = \emptyset$.
We assert that the integrity of a composite target is determined by the integrity of its active parties.

D6 $P(O_B^A{=}(k, n-k)|R_B{=}x) = \binom{n}{n-k} x^k (1-x)^{n-k} \lambda(n)$.
Assumes that the probability that A and B had an interaction history with size n is $\lambda(n)$, and that each past interaction had success probability x.

I1 For $W \in \mathbb{W}\backslash\{O_B^A\}$, it holds that $O_B^A \perp\!\!\!\perp W|R_B$.
The interaction history is completely determined by its size, and the probability of a success in a single interaction (by Dependency D6).

I2 For $W \in \mathbb{W}\backslash\{E_S : A \in \mathrm{act}(S), E_S \in \mathbb{E}\}$, it holds that $E_A \perp\!\!\!\perp W|R_A$.
The behavior of A is completely determined by its integrity parameter (by Dependency D2).

I3 For $W \in \mathbb{W}\backslash\{R_B\}$, it holds that $R_B \perp\!\!\!\perp W|E_B \cap \bigcap_{C\in\mathbf{A}}\{O_B^C\}$.
The only indicators of the integrity parameter of B, are interactions with it.

Independency I2 can be generalized for composite targets.

Proposition 1. *For all $W \in \mathbb{W}\backslash\{E_S : \mathrm{act}(T) \cap \mathrm{act}(S) \neq \emptyset, R_S \in \mathbb{E}\}$, it holds that $E_T \perp\!\!\!\perp W|R_T$.*

Proof. Apply structural induction. The base case precisely matches Independency I2. For the induction step use, that by definition of $\mathrm{act}(\cdot)$, it holds that $\mathrm{act}(T) \cup \mathrm{act}(T') = \mathrm{act}(T \wedge T') = \mathrm{act}(T \vee T')$. □

Our assumption is that trust adheres to D1-D6 and I1-I3.

A trust opinion of A about T can now be seen as the probability density function given by $f_{R_T}(x|\varphi)$, where φ is a condition that represents all knowledge of A about T, modulo the relations of the random variables. Typically, φ is the intersection of O_C^A, for different agents $C \in \mathbf{A}$. In other words, subjects have interaction histories about a group of active parties. If we restrict φ to merely the interaction history about a single active party, we get a beta distribution representing a simple trust opinion.

Lemma 1. *The simple trust opinion obtained from an interaction history with m successes and n failures is the beta distribution $f_{\mathrm{B}}(x; m+1, n+1)$.*

Proof.

$$
\begin{aligned}
& f_{R_B}(x|O_B^A{=}(m,n)) \\
=& \frac{P(O_B^A{=}(m,n)|R_B{=}x)\cdot f_{R_B}(x)}{\int_0^1 P(O_B^A{=}(m,n)|R_B{=}x')\cdot f_{R_B}(x')\,\mathrm{d}x'} \\
=& \frac{\binom{m+n}{m}x^m(1-x)^n\lambda(m+n)\cdot f_{R_B}(x)}{\int_0^1 \binom{m+n}{m}(x')^m(1-x')^n\lambda(m+n)\cdot f_{R_B}(x')\,\mathrm{d}x'} \\
=& \frac{\binom{m+n}{m}x^m(1-x)^n}{\int_0^1 \binom{m+n}{m}(x')^m(1-x')^n\,\mathrm{d}x'} \\
=& f_{\mathrm{B}}(x;m+1,n+1).
\end{aligned}
$$

□

The beta model ([5] and [6]) is based upon the notion that simple trust opinions are beta distribution. We can imagine an operator, trust aggregation, that updates trust opinions by adding more interactions. Formally, if we have a trust opinion X based on interaction history $(x_{\mathrm{S}}, x_{\mathrm{F}})$ and a trust opinion Y based on interaction history $(y_{\mathrm{S}}, y_{\mathrm{F}})$, then the aggregate of X and Y is a trust opinion based on $(x_{\mathrm{S}} + y_{\mathrm{S}}, x_{\mathrm{F}} + y_{\mathrm{F}})$. As such, the beta model inherits the mathematical property that the set of beta distributions is closed under trust aggregation.

Our assumptions regarding simple trust opinions are in line with the beta model, and are in fact sufficient to derive it (as demonstrated in Lemma 1). Hence, those assumptions can be seen as valid for the numerous models based on the beta model [8,9,10]. We extend the assumptions about simple trust opinions, by adding assumptions about composite trust opinions (Dependencies D3, D4 and D5). Under these assumptions, we show in Theorem 5 that composite trust opinions cannot generally be represented as beta distributions.

4 Composite Trust

In Example 1, we introduced the CLOUD grid. An example of a composite target was $D \vee (B \wedge C)$, where B, C and D are providers. The subject, A, has a (potentially empty) interaction history about B, C and D. In Example 2, we formally derive the trust opinion of A.

Example 2. The subject wants to form a trust opinion about $D \vee (B \wedge C)$, using only the interaction history of A about active parties B, C and D. The random variables O_B^A, O_C^A and O_D^A represent the interaction history of A about B, C and D. The random variable $R_{D\vee(B\wedge C)}$ represents the (unknown) integrity parameter of the target $D \vee (B \wedge C)$, and the random variable $E_{D\vee(B\wedge C)}$ represents the (unknown) outcomes of the next interaction with the target $D \vee (B \wedge C)$. We are interested not just in the probability of the next outcome of the target is a success ($E_{D\vee(B\wedge C)}$), but also in additional information, i.e. the random variable $R_{D\vee(B\wedge C)}$. Figure 2 depicts the relation between the users and the involved random variables. As stated in Section 3, given failures and successes of past interactions $(b_{\mathrm{S}}, b_{\mathrm{F}}, c_{\mathrm{S}}, c_{\mathrm{F}}, d_{\mathrm{S}}, d_{\mathrm{F}})$, the query for the trust opinion is of the shape

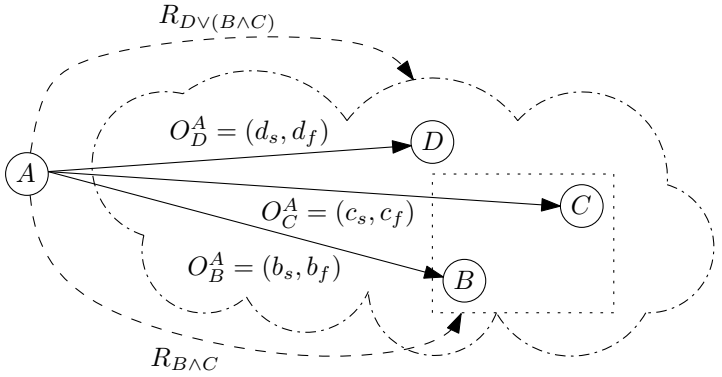


Fig. 2. Solid arrows represent interaction histories. Dashed arrows represent composite trust opinions. Arrows are labeled with the relevant random variables.

$f_{R_{D\vee(B\wedge C)}}(x|O_B^A = (b_{\mathrm{S}}, b_{\mathrm{F}}) \cap O_C^A = (c_{\mathrm{S}}, c_{\mathrm{F}}) \cap O_D^A = (d_{\mathrm{S}}, d_{\mathrm{F}}))$. In other words, the trust opinion represents the probability distribution of a random variable that predicts the probability that the target succeeds.

Whenever a subject wants to compute a composite trust opinion about a target, he chooses the correct conditions and the correct random variable to form a distribution over, as illustrated in Example 2. Therefore, we can assume, without loss of generality, that we are given the term representing the probability distribution, and we want to compute an explicit probability density function.

We are interested in a random variable R_T, where T is not a single agent (unless the subject wants a simple trust opinion). However, we have not provided direct relations between R_T and observation histories O_B^A or integrity parameters of single agents R_A. The only random variable that we can immediately relate R_T to is E_T. For more concise notation, we note the following lemma.

Lemma 2. *If S and T do not share any active parties, then $R_{S\wedge T} = R_S \cdot R_T$.*

Proof. The product $R_S \cdot R_T$ of two random variables is defined as $(R_S \cdot R_T)(\omega) := R_S(\omega) \cdot R_T(\omega)$.

By Dependency D2, it holds that

$$P(E_{S\wedge T} = \mathrm{S} | R_{S\wedge T} = x) = x.$$

And, using Proposition 1 as well as Dependencies D2 and D3 we obtain

$$\begin{aligned}
&P(E_{S\wedge T} | R_S = y \cap R_T = z) \\
=&P(E_S = \mathrm{S} \cap E_T = \mathrm{S} | R_S = y \cap R_T = z) \\
=&P(E_S = \mathrm{S} | E_T = \mathrm{S} \cap R_S = y \cap R_T = z) \cdot P(E_T = \mathrm{S} | R_S = y \cap R_T = z) \\
=&P(E_S = \mathrm{S} | R_S = y) \cdot P(E_T = \mathrm{S} | R_T = z) \\
=&y \cdot z.
\end{aligned}$$

Assume, without loss of generality, that $R_S(\omega) = y$ and $R_T(\omega) = z$. By Dependency D5, there is a function f such that $x = P(E_{S\wedge T} = \mathrm{s}|R_{S\wedge T} = x) = P(E_{S\wedge T} = \mathrm{s}|f(R_S, R_T) = x)$. That implies that $x = f(y, z)$, and thus $P(E_{S\wedge T} = \mathrm{s}|f(R_S, R_T) = f(y, z)) = f(y, z)$. Now, since $R_S(\omega) = y$ and $R_T(\omega) = z$, we have

$$\begin{aligned} & f(y,z) \\ =& P(E_{S\wedge T} = \mathrm{s}|f(R_S, R_T) = f(y,z)) = f(y,z) \\ =& P(E_{S\wedge T}) \\ =& P(E_{S\wedge T}|R_S = y \cap R_T = z) \\ =& y \cdot z. \end{aligned}$$

Thus $R_S \cdot R_T = f(R_S, R_T) = R_{S\wedge T}$. □

A similar proof exists for disjunction, using independency over union rather than intersection, yielding $R_{S\vee T} = R_S + R_T - R_S \cdot R_T$

We can derive the probability density function of $R_{S\wedge T}$ under any condition φ.

Theorem 4. *If S and T do not share any active parties, then*

$$f_{R_{S\wedge T}}(x|\varphi) = \int_x^1 \frac{1}{y} \cdot f_{R_S}(\frac{x}{y}|\varphi) \cdot f_{R_T}(y|\varphi)\,\mathrm{d}y.$$

Proof. Apply Theorem 3 and Lemma 2. It suffices to verify the integral bounds. $f_{R_S}(\frac{x}{y}|\varphi) = 0$ for $0 > \frac{x}{y}$ and $1 < \frac{x}{y}$, so we can ignore cases where $y < x$ and $y > 1$. □

The case for disjunction can be calculated in a similar fashion. Theorem 4 is sufficient to derive trust opinions about arbitrary targets (where no active parties appear more than once), given arbitrary interactions with the active parties.

Corollary 1. *For every (finite) target where no active parties appear more than once, an explicit function for the trust opinion can be computed by the subject.*

Proof. Apply structural induction over the shape of the target. The base case (simple trust opinions) is proven in Lemma 1. To prove the induction step, take Theorem 4 as a rewrite rule from left to right. □

In Example 3, we derive an explicit formula for the trust opinion of $B \wedge C$, and look at some of its properties.

Example 3. Assume that the subject, A, wants to establish a trust opinion about the target, $B \wedge C$. In the past, A has interacted as a passive party with B several times; five times B behaved well, and once badly. Furthermore, A has interacted with C, too; four times C behaved well, and twice badly. The trust opinion of A

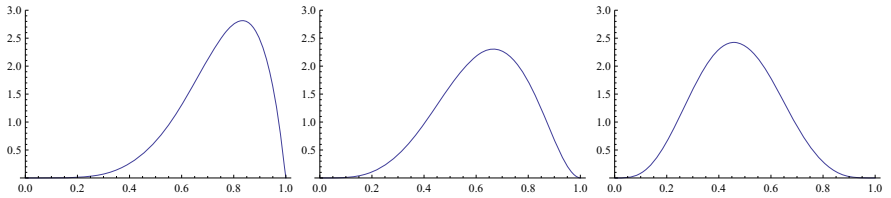


Fig. 3. From left to right: trust opinion about B, about C and about $B \wedge C$

about $B \wedge C$ is $f_{R_{B\wedge C}}(x|O^A_B = (5,1) \cap O^A_C = (4,2))$. Using Theorem 4, the trust opinion can be computed as

$$\int_x^1 \frac{1}{y} \cdot f_{R_B}(\frac{x}{y}|O^A_B = (5,1) \cap O^A_C = (4,2)) \cdot f_{R_C}(y|O^A_B = (5,1) \cap O^A_C = (4,2))\,\mathrm{d}y.$$

By Independency I3, we obtain

$$\int_x^1 \frac{1}{y} \cdot f_{R_B}(\frac{x}{y}|O^A_B = (5,1)) \cdot f_{R_C}(y|O^A_C = (4,2))\,\mathrm{d}y.$$

Which by Lemma 1 is equal to

$$\int_x^1 \frac{\frac{1}{y} \cdot (\frac{x}{y})^6 \cdot (1-\frac{x}{y})^2 \cdot y^5 \cdot (1-y)^3}{\mathrm{B}(5,1) \cdot \mathrm{B}(4,2)}\,\mathrm{d}y.$$

The formula can be formulated without an integral, and instead using some combinatorial functions, so that it reduces to

$$\frac{x^2 \cdot (1-x)^4 \cdot \Gamma(2) \cdot \Gamma(3) \cdot {}_2F_1(2,3;5;\frac{x-1}{x})}{\Gamma(5) \cdot \mathrm{B}(5,1) \cdot \mathrm{B}(4,2)}.$$

where Γ is the gamma function, B the beta function (not to be confused with the beta distribution) and ${}_2F_1$ a hypergeometric distribution. This, in turn, simplifies to

$$2205x^4(1 + 4x - 5x^2 + 2x(2+x)\log(x)).$$

The conjunction operation is depicted graphically in Figure 3. The rightmost distribution is the conjunction of the other two distributions. Recall that the abscissa depicts the integrity parameter of the targets in question. Thus, the more mass is on the right hand side of the graph, the bigger the probability that the target has a high integrity. As we can see, both active parties (B and C) have a relatively high integrity, but their conjunction ($B \wedge C$) does not.

The expected value of the trust opinion about a target is equal to the probability that the target succeeds, computation for $B \wedge C$ yields $\frac{15}{32}$. The expected value for the single agent B to succeed is $\frac{3}{4}$ and for C to succeed is $\frac{5}{8}$. Not coincidentally, the expected value for $B \wedge C$ is the product of that of B and C, namely $\frac{15}{32} = \frac{3}{4} \cdot \frac{5}{8}$.

The entropy of the trust opinion can be calculated by applying Definition 8 to our probability density function. This results in an entropy value of approximately -0.67685 bits. The entropy in the trust opinions for the single agents B and C is -0.86157 bits and -0.62058 bits, respectively. We can see that the amount of information we have about $B \wedge C$ is between the amount of information we have on B and on C. This does not generally hold. If we swap the successes and failures, the entropy for B and C does not change, but the entropy for $B \wedge C$ becomes -2.0811 bits. The reason for the difference in information is obvious, as conjunction is not symmetric with respect to the duality of successes and failures. For conjunction, a failure carries more information, since failures outweigh successes similar to how false overrules true in the logical conjunction.

As we suspected in Section 1, and seen for a specific case in Example 3, the expected behavior of a conjunction of targets, is equal to product of the expected behavior of both targets.

Corollary 2. *If S and T do not share any active parties, then*

$$\mathbf{E}(R_{S\wedge T}) = \mathbf{E}(R_S) \cdot \mathbf{E}(R_T).$$

Proof. Immediate consequence of Lemma 2. □

Although the derivation in Example 3 seems asymmetrical with respect to S and T, commutativity and associativity hold.

Corollary 3. *Conjunctions and disjunctions of independent trust opinions are commutative and associative.*

Proof. Immediate consequence of Lemma 2. □

In Example 3, we have shown a specific composite trust opinion to be

$$f_{R_{B\wedge C}}(x|O_B^A = (5,1) \cap O_C^A = (4,2)) = 2205x^4(1 + 4x - 5x^2 + 2x(2 + x)\log(x)).$$

Now, one can wonder whether there exists a beta distribution with a probability density function of that shape. It is important to realize that if (composite) trust opinions are closed under conjunction (and disjunction), then there must be such a beta distribution.

Theorem 5. *A composite trust opinion need not be representable by a beta distribution.*

Proof. The expression $2205x^4(1 + 4x - 5x^2 + 2x(2 + x)\log(x))$, is a composite trust opinion, but not a polynomial. The probability density function of a beta distribution is always a polynomial (see Definition 7). Hence that composite trust opinion is not based on a beta distribution. □

From Theorem 5, we can conclude that every trust model in which the trust opinions are (isomorphic to) beta models violates one of the assumptions. A famous example is Subjective Logic [8] (binomial, without base rate), other

examples include CertainLogic [11]. As the methodology of this paper is inspired by Subjective Logic, Dependencies D1, D2 and D6 are in line with the assumptions in Subjective Logic. Furthermore, the Independencies I1, I2, and I3 are also based on (non-formal formulations in) Subjective Logic. By the pigeon hole principle, Dependency D3 for conjunctions (or Dependency D4 for disjunctions) or Dependency D5 must be violated. Dependency D3 states that $E_{S \wedge T} = \mathrm{s}$ iff $E_S = \mathrm{s}$ and $E_T = \mathrm{s}$ (for independent S and T), and Dependency D5 asserts that the integrity of a composite target is determined by the integrity of the active parties. We believe that these assertions may not be considered erroneous. We do not propose to alter Subjective Logic, as one of the strong points of Subjective Logic is its simple representation (triples with belief, disbelief and uncertainty components), which is isomorphic to beta distributions. And, as proven in Theorem 5, we cannot adhere to all assumptions and have a representation of trust opinions isomorphic to beta distributions.

5 Conclusion

The paper makes several assumptions about the trust domain. The assumptions are designed having interactions over the Internet in mind. There, agents have trust opinions about other agents, and they update their trust opinions when new information becomes available. We argue that a trust opinion is not just an estimated integrity parameter of a target, but that a trust opinion is a probability distribution over the integrity of a target. The advantage is that the subject can derive much more than just the expected value from a probability distribution. Examples of additional key figures that can be deduced from a probability distribution are uncertainty (as entropy), confidence intervals, most probably integrity value (which does not usually equal the expected integrity), error margins (as variance) and the impact of new information (by updating the probability distribution with the new information).

The idea of using probability distributions over an integrity parameter is not new in the trust domain, as it was used in [5] and [6]. The novel idea is to not just use probability distributions over integrity as trust opinions, but to pick $f_X(x|I)$ as the probability distribution, where X is the target of the trust opinion, and I is the information the subject has. To get an explicit formula for the probability density function, we must introduce specific assumptions. The advantage of deriving the formula from these assumptions is threefold. First, by having explicit assumptions, any criticism on the resulting formula must be reducible to a disagreement about one of the assumptions. Second, if there is a disagreement about the assumptions, one can simply alter the assumption, and look at the implications. In particular, the assumption that all integrity parameters are equally likely for a simple target in the absence of information, is a strong assumption. The assumption can be replaced by asserting a different initial distribution of integrity parameters, and the model does not fundamentally change. Third, extending the formalism with new constructs may be achieved by adding new random variables and assumptions thereon.

An obvious candidate for extending the model is trust chaining, i.e. having the ability to use a recommendation as a (potential) source of information. If we extend the framework with trust chaining, we need to introduce random variables for (possible) recommendations, and introduce assumptions about when agents make honest recommendations and when they make dishonest recommendations. The suggested variants for trust chaining in different models are even more diverse than for conjunction and disjunction, partially due to different implicit assumptions and partially due to different insights [21]. Our approach may help unifying some insights, as well as force the assumptions to be formulated explicitly, thereby mitigating misunderstandings.

In this paper, however, we have applied the approach to composite trust opinions; trust opinions about conjunctions and/or disjunctions of agents. Thus, we have derived an explicit definition of a trust opinion of the shape "Can I trust that both A and B will behave according to agreement?" Of course, more general statements exist, where for "A and B" any propositional formula can be substituted and our result also generalizes to encompass these as well. We have proven some properties about composite trust opinions. First, the trust opinion about a target $S \wedge T$ has the expected value $s \cdot t$, where s and t are the expected values of the trust opinion about S and T. (Similarly, for $S \vee T$, it is $s + t - s \cdot t$.) Second, a composite trust opinion is in general not a beta distribution. Hence, no trust model with elements isomorphic to beta distributions can satisfy all our assumptions.

References

1. Castelfranchi, C., Falcone, R.: Principles of trust for MAS: Cognitive anatomy, social importance, and quantification. In: Proceedings of the 3rd International Conference on Multi Agent Systems. ICMAS '98, IEEE Computer Society (1998) 72–79
2. Rousseau, D.: Not so different after all : A cross-discipline view of trust. Academy of Management Review **23**(3) (1998) 393–404
3. Nooteboom, B.: Trust: Forms, Foundations, Functions, Failures and Figures. Edward Elgar (2002)
4. Anderson, D.P.: BOINC: A System for Public-Resource Computing and Storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing. GRID '04, Washington, DC, USA, IEEE Computer Society (2004) 4–10
5. Jøsang, A., Ismail, R.: The beta reputation system. In: Proceedings of the 15th Bled Electronic Commerce Conference. Volume 160., Citeseer (2002) 324–337
6. Mui, L., Mohtashemi, M.: A computational model of trust and reputation. In: Proceedings of the 35th Hawaii International Conference on System Science (HICSS). (2002)
7. Johnson, N.L., Kotz, S., Balakrishnan, N.: Beta Distributions. In: Continuous Univariate Distributions. 2 edn. Volume 2. Wiley (1995)
8. Jøsang, A.: Artificial reasoning with subjective logic. In: 2nd Australian Workshop on Commonsense Reasoning. (1997)
9. Teacy, W., Patel, J., Jennings, N., Luck, M.: TRAVOS: Trust and Reputation in the Context of Inaccurate Information Sources. Autonomous Agents and Multi-Agent Systems **12** (2006) 183–198

10. Ries, S.: Certain trust: a trust model for users and agents. In: Proceedings of the 2007 ACM symposium on Applied computing. SAC '07, ACM (2007) 1599–1604
11. Ries, S., Habib, S., Mühlhäuser, M., Varadharajan, V.: Certainlogic: A logic for modeling trust and uncertainty. In: Trust and Trustworthy Computing. Volume 6740 of Lecture Notes in Computer Science. Springer (2011) 254–261
12. George J. Klir, B.Y.: Fuzzy sets and fuzzy logic: Theory and applications. Upper Saddle River, New Jersey : Prentice Hall (1995)
13. Dempster, A.P.: Upper and lower probabilities induced by a multivalued mapping. Ann. Math. Statist. **38**(2) (1967) 325–339
14. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press (1976)
15. Jøsang, A., Haller, J.: Dirichlet Reputation Systems. In: International Conference on Availability, Reliability and Security, Los Alamitos, CA, USA, IEEE Computer Society (2007) 112–119
16. ElSalamouny, E., Sassone, V., Nielsen, M.: HMM-Based Trust Model. In: Formal Aspects in Security and Trust. Volume 5983 of LNCS. Springer (2010) 21–35
17. Billingsley, P.: Probability and measure. 3 edn. Wiley (1995)
18. Gut, A.: Probability: A Graduate Course (Springer Texts in Statistics). Springer (April 2007)
19. McEliece, R.J.: Theory of Information and Coding. 2 edn. Cambridge University Press (2001)
20. McCabe, K.A., Rigdon, M.L., Smith, V.L.: Positive reciprocity and intentions in trust games. Journal of Economic Behavior & Organization **52**(2) (2003) 267 – 275
21. Jøsang, A., Marsh, S., Pope, S.: Exploring different types of trust propagation. In: Trust Management. Volume 3986 of LNCS. Springer (2006) 179–192

IPv6 Stateless Address Autoconfiguration: Balancing between Security, Privacy and Usability

Ahmad AlSa'deh, Hosnieh Rafiee, and Christoph Meinel

Hasso-Plattner-Institute at University of Potsdam,
Potsdam, Germany
{ahmad.alsadeh,hosnieh.rafiee,christoph.meinel}@hpi.uni-potsdam.de

Abstract. Included in the IPv6 suite is a method for devices to automatically configure their own addresses in a secure manner. This technique is called Cryptographically Generated Addresses (CGAs). CGA provides the ownership proof necessary for an IPv6 address without relying on any trust authority. However, the CGAs computation is very high, especially for a high security level defined by the security parameter (Sec). Therefore, the high cost of address generation may keep hosts that use a high Sec values from changing their addresses on a frequent basis. This results in hosts still being susceptible to privacy related attacks. This paper proposes modifications to the standard CGA to make it more applicable security approach while protecting user privacy. We make CGA more privacy-conscious by changing addresses over time which protects users from being tracked. We propose to reduce the CGA granularity of the security level from 16 to 8. We believe that an 8 granularity is more feasible for use in most applications and scenarios. These extensions to the standard CGA are implemented and evaluated.

Keywords: IPv6 Security, IPv6 Address Autoconfiguration, Users' Privacy.

1 Introduction

IPv6 comes with many enhancements to IPv4. One major enhancement is the stateless autoconfiguration feature, which allows a node to self-determine its own address without the need of a Dynamic Host Configuration Protocol (DHCP) server. The autoconfiguration feature enables nodes to directly connect to the network. The host uses the information advertised by the router and its interface identifier (IID) information to construct its address.

The most well-known way of setting the IID is based on Neighbor Discovery (ND) [1] and Stateless Address Autoconfiguration (SLAAC) [2]. SLAAC embeds a network devices Ethernet Media Access Control (MAC) address into an IPv6 address. Since every MAC address is unique, IPv6 could allow devices to be globally uniquely identified. Unfortunately this uniqueness property can allow for the tracking of an individuals device thus violating the users privacy. In practice, a lot of devices (e.g., laptops, cell phones, etc.) are associated with individual users.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 149–161, 2013.

IPv6 *privacy extensions* [3] help to protect users from being tracked. This technique is used to assign temporary IPv6 address values that change over time. Changing the addresses over time makes it more difficult for eavesdroppers to track nodes as they roam between networks. It is also harder to make a correlation between addresses when different addresses are used for different activities corresponding to the same node. Although the *privacy extensions* protect the users' privacy, they cannot prevent attacks related to IP spoofing.

Besides dealing with the privacy issues, ND and SLAAC are also directed at malicious operators. RFC 3756 [4] describes possible attacks against ND. The SEcure Neighbor Discovery (SEND) [5] was developed to counteract the vulnerabilities in ND and SLAAC.

SEND is mainly dependent on Cryptographically Generated Addresses (CGAs) [6]. CGA is a technique that offers the authentication of IPv6 addresses without the need of a third party or additional security infrastructure. In CGA addresses, the IIDs are generated by one-way hashing of the nodes public key and other auxiliary parameters such as *Modifier, Subnet Prefix* and *Collision Count*. Thus the IPv6 node address is bound to its public key.

The high computational cost is the main disadvantage for using CGA. It is likely that once a host generates an acceptable CGA, it will continue to use this address. This results in hosts using CGAs still being susceptible to privacy related attacks. It is therefore important to find a way to balance between security, privacy and usability issues of CGA before the wide deployment of IPv6 is undertaken.

IPv6 *privacy extensions* provide the privacy protection necessary for nodes in IPv6 networks but it cannot secure the addresses. While the CGA can prevent address theft related attacks, it is computationally heavy and might be vulnerable to privacy related attacks. Therefore, it is important to find an in between approach which offers security and protection of the users' privacy.

In this paper we define a mechanism that eliminates the security and privacy concerns in IPv6 SLAAC. To protect the users' privacy, we present a modified CGA implementation that integrates the *privacy extensions* approach into CGA. In this way, CGA will prevent IPv6 address spoofing related attacks while changing addresses over time will protect the users' privacy. We also propose to reduce the security level CGA granularity from 16 to 8, to better increase the chance of having a better security level (Sec) and avoiding the large step between the successive Sec values. We also modify the CGA implementation to allow for the generation of the public key pair on-the-fly to increase the randomness of CGA addresses and to enhance the users' privacy protection.

The paper is organized as follows. In Section 2, we briefly introduce the IPv6 Neighbor Discovery Protocol (NDP) and discuss its security and the privacy implications. In Section 3, we discuss the related work to solve the NDP privacy and security issues. In Section 4, we provide the details of the modifications we propose for CGA to achieve the security and privacy in feasible way. In section 5, we evaluate the implementation and usage of the modified CGA and discus the compatibility concerns and deployment limitations. In Section 6, we present our conclusions.

2 Neighbor Discovery Protocol (NDP)

ND for IPv6 [1] and IPv6 SLAAC [2] together are referred to as NDP. NDP is one of the main protocols in the IPv6 suite. It is heavily used for several critical functions, such as discovering other existing nodes on the same link, determining others link layer addresses, detecting duplicate addresses, finding routers and maintaining reachability information about paths to active neighbors.

2.1 Stateless Address Autoconfiguration (SLAAC)

In IPv6 SLAAC, the node creates the rightmost 64 bits (IID), which identifies an individual node within a local network. The IID is often configured from the Extended Unique Identifier (EUI-64) that is generated based on the interface hardware identifier - usually the MAC address of the network card. Afterwards, the node combines the subnet prefix with the IID to form a complete 128 bits IPv6 address. The subnet prefix can be the reserved local link prefix used to generate local link addresses or the prefix which is advertised by the router through the Router Advertisement (RA) message. Finally, the Duplicate Address Detection (DAD) algorithm is run by the node to ensure that there is no address conflict on the same link. The IID includes two bits which are reserved by the IPv6 addressing architecture for special purpose. The 7th bit from the left in the 64-bit IID is the Universal/Local bit (u bit). The 8th bit from the left is the Individual/Group (g bit).

2.2 SLAAC Privacy Implications

IPv6 SLAAC leads to serious privacy implications because IPv6 address may reveal sufficient information to identify the hardware and track the individual. Generating an IID based on the MAC address results in a static IID. This IID will remain the same same in all networks the node contacts. This means that the host MAC address is exposed to the Internet because any website a user visits will log his IP. Consequently, an attacker can use data mining techniques to correlate the users activity based on the IID. A more worrying case concerns mobile devices (e.g., cell phones, laptops, PDAs, etc.) because most of these devices are associated with individual users. An attacker can use the IID to track the movement and the usage of a particular device. Once the location and the identity of the user are determined, an attacker can target the user for identity theft or other related crimes [7].

2.3 NDP Vulnerabilities

It is easy to perform attacks against NDP because it has no authentication mechanism. For instance, it is difficult for a node to distinguish between a fake and the authorized routers' advertisements. The newly connected node cannot validate the routers before having an IP address. Thus, a malicious node can send a fake RA to perform Denial-of-Service (DoS) or Man-in-the-Middle (MitM)

attacks and effectively receive, drop, or replay the packets. An attacker can also generate DoS on DAD to prevent a node from obtaining a network address. A malicious node may block the legitimate node from getting a new IPv6 address by always responding to every DAD attempt with the spoofed message that "I have this address". The victim would thus find out that every IPv6 address that it tried to use was being used by other nodes. It would therefore be unable to obtain an IP address to access the network. RFC 3756 [4] describes the possible attacks against NDP.

3 Approaches to Mitigate NDP Privacy and Security Implications

3.1 Privacy Extensions for SLAAC in IPv6

IPv6 *privacy extensions* [3] describe a technique for assigning temporary IPv6 addresses that change over time. Changing the addresses over time makes it more difficult for eavesdroppers and other information collectors to correlate IP address with host (user) when different addresses used for different activity correspond to the same host. The random IID is generated via a hash function using a quantity which forces randomization of the IID. A node can use different IIDs with different prefixes to have a set of global addresses that cannot be easily linked to each other. The temporary address would be used for a certain period of time and then would be deprecated.

Although the *privacy extensions* can protect a users' privacy it cannot prevent IP spoofing related attacks. The privacy extensions have no authentication mechanism with which to enable the receiver to verify the identity of the sender. Therefore, an attacker can usurp other users' addresses to carry out a wide variety of attacks. Fortunately another approach which is called CGA [6] can provide the authentication needed to prevent address theft in the IPv6 environment.

3.2 Cryptographically Generated Addresses (CGAs)

CGA Generation Algorithm. In CGA, the IID portion of IPv6 address is created from a cryptographic hash of the address owner's public key and other auxiliary parameters - *Modifier*, *Collision Count* and *Subnet Prefix*. The address owner computes two hash values (Hash1 and Hash2). The combination of the two hash values increases the computational complexity for the attacker to do the brute-force search attack. Since the 64-bit are not enough to provide sufficient security against brute-force attacks in the foreseeable future, the standard CGA uses the *Hash Extension* (Hash2) to increase the security strength above 64-bit. The computational complexity of Hash2 depends on the Sec value. Sec is an unsigned 3-bit integer having a value between 0 and 7 which indicates the security level of the generated address.

Each CGA is associated with a CGA parameters data structure, which contains the following fields:

- *Modifier* (128 bits): it is initialized to a random value.
- *Subnet Prefix*(64 bits): it is set to the routing prefix value advertised by the router at the local subnet.
- *Collision Count* (8 bits): it is the result of a collision counter used for DAD algorithm to ensure the uniqueness of the generated address.
- *Public Key* (variable length): it is set to the DER-encoded public key of the address owner.
- *Extension Field* has a variable length for future needs.

Fig. 1 shows a schematic of the CGA generation algorithm. CGA generation begins with determining the address owner's public key and selecting the proper Sec value. The Hash2 computation loop then continues until finding the final *Modifier*. The Hash2 value is a hash of the combination of the *Modifier* and the *Public Key* which are concatenated with a zero-value for the *Subnet Prefix* and the *Collision Count*. The address generator tries different values of the *Modifier* until $16\times Sec$-leftmost bits of Hash2 become zero. Once a match is found, the loop for the Hash2 computation terminates. Then the final *Modifier* value is saved and used as an input for the Hash1 computation. The Hash1 value is a hash of the combination of the whole CGA parameters. The IID is then derived from Hash1. The Sec value is encoded into the three leftmost bits of the IID. Finally, the DAD algorithm is run by the client to ensure that the address is unique within the same subnet. If an address collision occurs, increment the *Collision Count* and compute Hash1 again to get the IID. However, after three collisions, CGA algorithm stops and reports an error.

To assert the ownership of the address and to protect the message, the address owner uses the private key that corresponds to the *Public Key* in the CGA parameters to sign messages sent from that address. Finally, the node will send the message, the CGA parameters, and the signature.

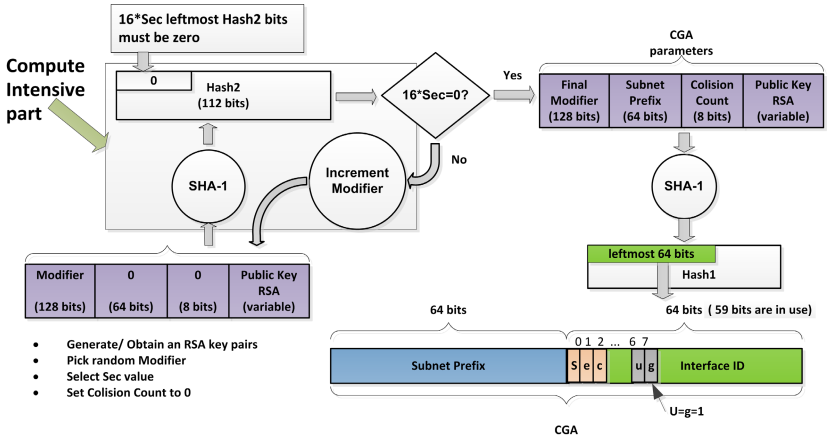


Fig. 1. CGA Generation Algorithm

CGA verification takes as input an IPv6 address and CGA parameters. If the verification succeeds, the verifier knows that the public key belongs to that address. Then, the verifier uses the public key to authenticate the signed messages from the address owner.

The CGA algorithm increases the computational cost for both the attacker and the address generator (owner). The address generator needs $O(2^{16\times Sec})$ brute-force search to satisfy the Hash2 condition and for finding the final *Modifier*. The attacker needs to do a brute-force attack against an $(16 \times Sec + 59)$-bit hash value which costs $O(2^{16\times Sec+59})$. Fulfilling the condition of Hash2 is the computationally expensive part of the CGA generation. Selecting a high Sec value may cause unacceptable delay in address generation. Even there is a probabilistic guarantee that the CGA address generation will stop after a certain number of iterations, but it is impossible to tell exactly how long it will take for the CGA generation when Sec is not zero.

CGA Privacy Concerns. With CGA, the *Modifier* is used to enhance the privacy by adding randomness to the address. Changing the *Modifier* over time leads to different IIDs. Therefore, CGA can provide IPv6 addresses with the privacy they need.

However, there are two apparent limitations to this privacy protection. First, hosts that use a high Sec value may choose not to change their addresses frequently. Due to the high computational complexity of generating Hash2, it is likely that once a host generates an acceptable CGA it will continue to use this fixed IID in multiple activities thus reducing its need for frequent regeneration - at least for that subnet. The result is that hosts using CGAs are still susceptible to privacy related attacks. Second, the *Public Key* of address owner is attached with message that is sent to the receiver. This means that the node can still be identified by its public key.

Therefore, the CGA has a privacy implication (especially for high Sec value) and the *privacy extensions* approach is vulnerable to the address spoofing related attacks. In the next section we will integrate the two approaches in a balanced way to attain both the security and the privacy in a usable method.

4 Modifications to Standard CGA

We propose three main modifications to the standard CGA process to attain the users' privacy in a practical manner. First, we modify the CGA to have a lifetime that indicates how long the address is bound to an interface. Second, we reduce the granularity of CGA security levels to get more practical security levels. Third, we generate the keys, on-the-fly, using CGA code to ensure more security and privacy for the users of CGA addresses.

4.1 Setting a Lifetime for Temporary CGA Addresses

We propose to change the CGA addresses periodically to protect the users privacy. Each CGA address has an associated lifetime that indicates how long the

address is bound to an interface. Once the lifetime expires, the CGA address is deprecated. While a CGA address is in a deprecated state, its use is discouraged, but not strictly forbidden. New communication (e.g., the opening of a new TCP connection) should use a new CGA address when possible. A deprecated address should be used only by applications that have been using it and would have difficulty switching to another address without a service disruption. When the lifetime expires and the address is not used by an opened connection, the CGA address is removed from the network interface by the kernel and no longer used.

The lifetime of a temporary CGA address depends on several parameters and actions. For instance, the lifetime should depend on the time needed for a host to generate a new CGA address, the time needed for an attacker to break the CGA address and user desired setting for security and privacy. The following lists the conditions under which a new temporary CGA address should be generated:

- When a host joins new subnet. In this case, the new CGA parameters will be used to generate the new address. A new public key will be used for calculating both the Hash1 and Hash2 values. In the standard CGA it is not necessary to use new CGA when the node moves to new subnet.
- Before the lifetime for the in-use CGA address has expired. To ensure that the CGA address is always available and valid, new CGAs should be regenerated in advance before the predecessor will be deprecated. In practice, a valid lifetime should not be zero. Using the standard *privacy extensions*, the default interval is 24 hours; however, we recommend a minimum lifetime of one hour.
- When the subnet prefix lifetime has expired. A new CGA address will then need to be regenerated. It must include the newly received prefix used in calculating Hash1.
- When the user needs to override the default value of the lifetime in order to generate a new CGA address. The CGA implementation should offer the user the ability to override the current lifetime values and force the CGA algorithm to generate a new address.

Determining the proper lifetime for a CGA address depends on the privacy and security level constraints. For the security level analysis we refer to the security model which has been proposed by Bos et al. [8] for studying the security and efficiency of the CGA. The necessary notation used in the CGA time analysis is defined as follows:

- T_G: The average time needed for a node to generate a CGA.
- T_A: The average time needed for an attacker to impersonate an address.
- T_1: The time needed to compute Hash1.
- T_2: The time needed to compute Hash2.
- b: The number of available bits in the address, which is the truncated output of Hash1 (IID).
- g: The granularity of the security level in CGA.
- s: The number of bits needed to satisfy the Hash2 condition ($s{=}g \times Sec$), which is the truncated output of Hash2.

The address generator needs on average $(2^s \times T_2)$ in order to fulfill the Hash2 condition, plus T_1 to generate the IID from Hash1. Therefore, the cost of address generation, T_G, is:

$$T_G = (2^{g \times Sec} \times T_2) + T_1 \tag{1}$$

An attacker has two ways to impersonate a node - by satisfying the constraints on Hash1 and then the conditions on Hash2 or vice versa. Beginning with Hash1, the attacker must first perform the attack on Hash1, which takes $(2^b \times T_1)$ hash function evaluations. Once fulfilled, the conditions on Hash2 for the generated *Modifier* should be satisfied, which takes 2^s hash function evaluations. Thus, the total time for impersonation when beginning with Hash1 (H_1) becomes T_A : $H_1 = (2^b \times T_1 + T_2)2^s$.

When the attacker starts from Hash2, the conditions on Hash2 are met at a cost of $(2^s \times T_2)$ hash function evaluations. Next, Hash1 is verified if it matches the target address. Hash1 verification costs 2^b. Therefore, the total cost when beginning with Hash2 (H_2) becomes $T_A : H_2 = (2^s \times T_2 + T_1)2^b$.

The attacker can choose between the two ways to minimize his attack cost. Hence, the time for impersonation an address (T_A) is:

$$T_A = min\left\{(2^{59} \times T_1 + T_2)2^{g \times Sec}, (2^{g \times Sec} \times T_2 + T_1)2^{59}\right\} \tag{2}$$

The resistance of CGA against impersonation is mainly controlled by increasing the number of bits on the Hash2 condition $s{=}g \times Sec$. For the standard CGA, with *g=16* and Sec value between 0 and 7, the number of operations required for impersonation on a specific node is:

$$T_A = \begin{cases} 2^{59} \times T_1 & \text{if } Sec = 0, \\ (2^{59} \times T_1 + T_2)2^{16 \times Sec} & \text{if } 1 \leq Sec \leq 3, \\ (2^{16 \times Sec} \times T_2 + T_1)2^{59} & \text{if } 4 \leq Sec \leq 7, \end{cases} \tag{3}$$

In next subsection, we propose to reduce the granularity of CGA for practical and usability reasons. When the granularity, g, is ≤ 8 the cost of address generation T_G becomes:

$$T_G = \Big\{(2^{8 \times Sec} \times T_2) + T_1 \quad \text{if } 0 \leq Sec \leq 7 \tag{4}$$

And the number of operations required for the impersonation of a specific node becomes:

$$T_A = \begin{cases} 2^{59} \times T_1 & \text{if } Sec = 0, \\ (2^{59} \times T_1 + T_2)2^{8 \times Sec} & \text{if } 1 \leq Sec \leq 7. \end{cases} \tag{5}$$

We can surmise from equation 5, that it would be easier for the attacker to start by calculating Hash1 then fulfilling the Hash2 condition. Here, the assumption is that the hash function has no known weaknesses.

The lifetime of a CGA address (T_l) should be safe enough so the attacker is not able to impersonate the other nodes' addresses. We recommend that T_A be

at least nT_l (where n is an integer) in order to have a safe margin. Clearly, the speed of hash function computation depends on the CPU speed of the computing device. Reading the CPU speed by using the CGA code makes it possible to determine whether or not the selected lifetime is suitable. On the other hand, the T_l time should be greater than the time required for the node to generate a CGA address. It is not feasible to invest the time and resources of the computing device to create an address and then, after a very short period of time, deprecate this address. We recommend that T_l be greater than mT_G (where m is an integer). Therefore, T_l can be described by the following equation:

$$mT_G \leq T_l \leq \frac{T_A}{n} \tag{6}$$

Where m and n are integers.

4.2 Reducing the Granularity of CGA Security Levels

In the CGA generation algorithm, the granularity factor 16 is relatively large. The multiplier 16 was chosen to increase the maximum length of the *Hash Extension* [6] up to 112 bits, but the benefit of this is questionable [9]. Currently, Sec value 0 or 1 can be used in practice. For Sec value 2, the CGA address generation process may take several hours or days. We carried out a test on a set of 5 samples using 2.67 GHz CPU speed which gave us an average CGA generation time of 5923857 Milliseconds (1 hour and 39 minutes). The CGA computation for a Sec value of 3 will take, on average, more than 12 years on a 2.67 GHz CPU.

Smaller granularity is more suitable for CGA computations. Therefore, we proposed to reduce the granularity factor from 16 to 8 for the following reasons:

- The granularity factor 16 is quite large and causes a big jump in CGA computation time for successive Sec values. Having values in between is better than waiting for a very long time to reach the second security level (Sec+1). A smaller granularity factor gives the users the opportunity to have better security level particularly if the user is not willing to wait a long time for the CGA generation. Having a Sec value of 1 with a granularity factor of 8 is better than a Sec value of 0 with a granularity factor of 16.
- Changing the CGA addresses over time in order to protect the users' privacy makes it unnecessary to select a high security level. It does not make sense for the address owner to select a high Sec value that is expensive in time and CPU cycles if the address will be changed after a short period of time. For instance, it is not reasonable to select a high Sec value which costs the address owner several days if the address will be changed every one hour due to the privacy need. However, the security level should be sufficient to cover a lifetime period. For example, if the lifetime is one day, the security level should be safe enough so that the attacker cannot break the address within several days.

- The privacy concerns are usually much more important for mobile devices. The mobile devices generally have limited resources (battery, memory, and processing power). For a high Sec value, the CGA computation will take too long a time and will consume too much of the computing device's energy. Smaller granularity is more suitable for these devices.
- The multiplication factor of 8 increases the maximum length of the *Hash Extension* up to 56 bits. Therefore, the total hash length will be between 59 and 115 bits, which are adequate for current CPU speeds. Decrementing the granularity to 4 or 2 might lead to weaker hash values which leave small margin of safety. With granularity 4, the total hash length will be between 59 and 87 bits.

4.3 Automatic Key Pair Generation

We propose to generate the key pairs automatically by using the CGA code as proposed in [10]. The default key size is 1024 bits. This can be changed by the use of the CGA parameter setting interface. Setting the keys automatically is better for the following reasons:

- Generating the key pair on-the-fly each time the host needs a new address enhances the CGA security and protects the user's privacy. The automatic generation of the public key increases the randomness of the CGA and consequently enhances its security against a brute-force attack. Moreover, each time the node moves to a new location, it will get a new CGA address and will use a new public key. Therefore, it will not be easy for attackers to track users based on their addresses or even to correlate traffic to their public keys.
- Minimizing the amount of required configurations so that the end user does not need to know the technical details behind the cryptography. There is also no need to use an external program to generate the key pairs. It is not easy for the user to generate key pairs manually each time the host wants to generate a new address. It becomes more tedious when the CGA address changes frequently due to privacy time constrains.
- Keys are not stored in a particular path before starting the CGA computation. The keys are therefore not vulnerable to theft. In our CGA implementation, the key pairs are stored in computer memory (RAM) for quick accessibility, but also the digested keys are stored in an XML file for further usage, such as after rebooting the system while the IP address is still valid or the host is connected to the same subnet.
- The average time to generate a key pair with a RSA 1024-bit key using 1000 samples is 27.8 Milliseconds, while the average time for Standard CGA generation with Sec value 1 and a 1024-bit key size is 439.6 Milliseconds. So, the key generation takes about 6.3% of the total CGA generation time. We took these measurements on a computer with 2.67 GHz.

5 Modified CGA Implementation and Its Evaluation

5.1 Modified CGA Implementation

To test the above mentioned modifications, we modified the CGA part of SEND implementation for the Windows operating system (WinSEND)[11]. WinSEND works as a service to provide security for Windows NDP. WinSEND has the SEND functionalities and can generate IP addresses in a secure manner.

Our modified version of CGA automatically offers the default parameters to generate a temporary CGA address. The default value for the minimum lifetime is 24 hours, similar to the proposed preferred lifetime in RFC 4941, and the public key size is 1024-bit. The default value for Sec is 2 and the granularity is 8. The user can override these parameters via CGA setting (See Fig. 2).

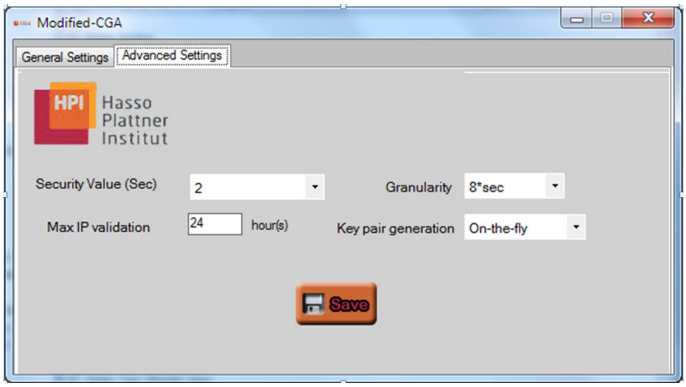


Fig. 2. Modified CGA settings parameters

To get a rough idea about the time required for generating a CGA address for different Sec values with associated granularity factors, we used the results shown in Table 1. The CGA generation algorithm is run on a 2.67 GHz Quad core CPU computer. The results are taken over 1000 samples with a 1024-bit key size.

Table 1. CGA Generation Time in Milliseconds (ms) for Different Sec Values with Different Granularity Over 1000 Samples

	Granularity		
Sec	4	8	16
1	117ms	121ms	427ms
2	128ms	425ms	5923857ms
3	135ms	88217ms	*
4	409ms	*	*

5.2 Limitations and Deployment Considerations

Our proposed modification to the standard CGA is compatible with addressing scheme and could be implemented as an extension to RFC 3972. The CGA-enabled nodes need to consider the granularity factor 8 in CGA generation and verification algorithms. This task is not complicated, eventually feasible modifications can be upgraded. The other modification for changing the addresses over time and generating keys on-the-fly do not affect the CGA algorithm and the way of communication. It is more implementation decisions which do not change the CGA algorithm.

There are some implications and deployment considerations for changeable addresses. Most of these limitations are also valid for the privacy extensions approach (RFC 4941) as explained below:

- The changeable address may cause unexpected difficulties with some applications. Some servers reject the connection from clients whose address cannot be mapped into a DNS name that also maps back into the same address.
- Changing the addresses frequently (e.g., every few minutes) has a performance implication and will severely impact user experience.
- Protecting the user's privacy may conflict with the administrative need to effectively maintain and debug the network.
- The implementation needs to keep track of the addresses being used by the upper layer in order to be able to remove the deprecated addresses from the internal data structure when these addresses are no longer used by the upper protocols, but not before.

Tracking users at other layers, such as tracking through DNS, cookies, or browser characteristics is out of the scope of this paper. However, in order to have privacy protection at higher-layers, we believe that the underlying protocols must also have privacy protection mechanisms.

6 Conclusion

It is very important to be sure that the increasing deployment of IPv6 will be done in a secure way without compromising the Internet users' privacy. It is proposed that CGA be used to prove the ownership of an IPv6 address and to prevent spoofing of existing IPv6 addresses, but it might be susceptible to privacy related attacks. On the other hand, the *privacy extensions* protect the users' privacy but are of no value to related address spoofing attacks. In this paper we showed how to integrate the *privacy extensions* into CGA to resolve both privacy and security issues for IPv6 addresses. We also changed the granularity of the CGA security level and generated the public-key pair on-the-fly to make CGA more practical. We also provided a mechanism for the CGA implementation with which to automatically set the maximum lifetime and to decrease administrative tasks. This approach definitely involves tradeoffs between privacy, security, usability and the cost of address generation but it is a very viable solution.

Our proposal has several benefits over the current CGA scheme, including: (1) the ability to diminish the CGA possible privacy concerns and protect users from being tracked; (2) the ability to configure when new CGA should be created; and (3) the possibility to have finer granularity for CGA security level. We have implemented and tested the CGA modification and found that it generates new CGA address as designed while not impacting Internet activities.

References

1. Narten, T., Nordmark, E., Simpson, W., Soliman, H.: Neighbor Discovery for IP version 6 (IPv6). RFC 4861, Internet Engineering Task Force (September 2007)
2. Thomson, S., Narten, T., Jinmei, T.: IPv6 Stateless Address Autoconfiguration. RFC 4862, Internet Engineering Task Force (September 2007)
3. Narten, T., Draves, R., Krishnan, S.: Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941, Internet Engineering Task Force (September 2007)
4. Nikander, P., Kempf, J., Nordmark, E.: IPv6 Neighbor Discovery (ND) Trust Models and Threats. RFC 3756 (Informational), Internet Engineering Task Force (May 2004)
5. Arkko, J. (ed.), Kempf, J., Zill, B., Nikander, P.: SEcure Neighbor Discovery (SEND). RFC 3971, Internet Engineering Task Force (March 2005)
6. Aura, T.: Cryptographically Generated Addresses (CGA). RFC 3972, Internet Engineering Task Force, updated by RFCs 4581, 4982 (March 2005)
7. Groat, S., Dunlop, M., Marchany, R., Tront, J.: The privacy implications of stateless IPv6 addressing. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, CSIIRW 2010, pp. 52:1–52:4. ACM, New York (2010)
8. Bos, J.W., Özen, O., Hubaux, J.-P.: Analysis and optimization of cryptographically generated addresses. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 17–32. Springer, Heidelberg (2009)
9. Alsa'deh, A., Rafiee, H., Meinel, C.: Stopping Time Condition for Practical IPv6 Cryptographically Generated Addresses. In: 2012 International Conference on Information Networking (ICOIN), pp. 257–262 (2012)
10. Rafiee, H., Alsa'deh, A., Meinel, C.: Multicore-based Auto-scaling SEcure Neighbor Discovery for Windows Operating Systems. In: 2012 International Conference on Information Networking (ICOIN), pp. 269–274 (2012)
11. Rafiee, H., AlSa'deh, A., Meinel, C.: WinSEND: Windows SEcure Neighbor Discovery. In: 4th International Conference on Security of Information and Networks (SIN 2011), Sydney, Australia, November 14-19, pp. 243–246. ACM (2011)

Policy Administration in Tag-Based Authorization

Sandro Etalle[1,2], Timothy L. Hinrichs[3], Adam J. Lee[4], Daniel Trivellato[1], and Nicola Zannone[1]

[1] Eindhoven University of Technology
[2] University of Twente
[3] University of Illinois at Chicago
[4] University of Pittsburgh

Abstract. Tag-Based Authorization (TBA) is a hybrid access control model that combines the ease of use of extensional access control models with the expressivity of logic-based formalisms. The main limitation of TBA is that it lacks support for policy administration. More precisely, it does not allow policy-writers to specify administrative policies that constrain the tags that users can assign, and to verify the compliance of assigned tags with these policies. In this paper we introduce TBA^2 (Tag-Based Authorization & Administration), an extension of TBA that enables policy administration in distributed systems. We show that TBA^2 is more expressive than TBA and than two reference administrative models proposed in the literature, namely HRU and ARBAC97.

Keywords: access control, policy administration, auditing.

1 Introduction

Access control systems in real-world organizations are mostly based on extensional approaches to access control (e.g., access control lists), as their ease of use is preferred to the flexibility of logic-based models. Authorization policies in extensional models are based on simple assignments of rights to users, or on the characterization of users in terms of properties (e.g., roles) and the assignment of rights based on those properties. Nevertheless, the lack of expressiveness of extensional models severely limits the constraints that can be expressed in authorization policies. A hybrid approach to access control that combines the usability of extensional models and the flexibility and expressiveness of logic-based formalisms would offer great benefits for the deployment of access control systems in real-world organizations.

To accommodate this need, Tag-Based Authorization (TBA) has been proposed by Hinrichs et al. [15], based on the work by Najafian Razavi and Iverson [20] and Wang et al. [26]. TBA is a hybrid access control model that relies on formal logic for the definition of authorization policies, and on extensional models for describing a system's subjects and objects in terms of simple properties (e.g., roles). This integration allows relatively untrained users to choose descriptive tags for the system's subjects and objects; security experts then write logical policies that define access authorizations using combinations of those tags. The resulting access control model is flexible and easy to use, yet expressive enough to match the needs of complex application domains.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 162–179, 2013.

As an example application domain for the TBA model, consider Operation Atalanta, a military operation involving several EU navies that collaborate to prevent criminal activities (such as smuggling and pirate attacks) off the Somali coast. The information gathered and exchanged by the collaborating vessels is typically processed and classified by the vessels' operators (e.g., a transiting ship is marked as suspicious). Yet, the policies for accessing this information are regulated by complex context- and content-based conditions and must be written by higher-rank personnel. By explicitly distinguishing the task of characterizing information and the task of writing the authorization policy, TBA leads to a better separation of duties and valorization of the skills of a vessel's workforce than the existing access control models.

If on the one hand allowing low-rank users to assign tags to subjects and objects greatly enhances the usability of the access control system, on the other hand it enables low-rank users to influence the system's authorizations. In fact, inaccurate tags (whether by intention or accident) can circumvent the intended authorization policy of the system. The main limitation of TBA in this respect is the lack of support for policy administration. In particular, TBA does not allow policy-writers to: (1) define which users may assign which tags to which subjects and objects, i.e., to specify *administrative policies*; (2) hold users accountable for the tags they assign, and verify the compliance of tags with administrative policies; and (3) revoke incorrect tags or tags assigned by unauthorized users. In the scenario above, for instance, it is important to allow only operators with appropriate clearance to tag sensitive information, and to verify whether the assigned tags comply with this policy (possibly revoking the tags that do not satisfy the policy).

In a centralized system, these issues can be addressed by means of a traditional access control mechanism that regulates the tagging process and a logging mechanism that records users' actions. In a distributed system, however, these solutions are insufficient as they would require entities in one security domain to trust the enforcement and logging mechanisms of systems in different security domains. Furthermore, tracing the objects exchanged in a distributed system might be difficult, complicating or even preventing the revocation process.

In this paper, we extend the TBA model from [15] to enable policy administration in distributed systems. In particular, we introduce TBA^2 (Tag-Based Authorization & Administration), an extension of TBA that allows security administrators to specify constraints on the tagging process and to verify the compliance of tags with administrative policies by combining traditional "a priori" access control mechanisms with "a posteriori" verification. Technically, our extension consists of associating an *issuer* to each tag, which identifies the user who assigned the tag. For example, all the tags issued by an operator will be marked with the identifier of the operator. In contrast to previous work on a posteriori verification (e.g., [7]), this enables policy compliance verification without the need of an auditing infrastructure, by exploiting the observability of users' actions (i.e., "signed" tags). As a consequence, TBA^2 represents a lightweight solution for the enforcement of authorization and administrative policies in distributed systems. We show that the proposed model is more expressive than the original TBA model and than two well-known administrative models proposed in the literature, namely the Harrison, Ruzzo, and Ullman model [13] and ARBAC97 [23].

The paper is organized as follows. Section 2 reviews the TBA model and discusses its limitations. Section 3 introduces TBA^2, and Section 4 evaluates its expressive power. Section 5 discusses related work, and Section 6 concludes the paper.

2 Background

Tag-Based Authorization (TBA) [15] is a hybrid access control model combining the flexibility and expressiveness of logic-based formalism with the ease of use of extensional models. In this section we first present the definitions of the TBA model given in [15] that are relevant for this paper, and then we identify the main limitations of TBA with respect to its deployment in a distributed system.

2.1 The TBA Model

In this paper we use S to denote the set of subjects, O to denote the set of objects, and R to denote the set of rights that can be assigned within a system. T denotes the set of possible tags that can be assigned to S and O, and $\mathsf{tag} : S \cup O \rightarrow 2^T$ is a function that maps a subject or object to the set of tags assigned to it. Tag denotes the set of all possible tag functions. A TBA authorization policy is written in some logical access control language $\langle P, L, \models \rangle$ defined as follows.

Definition 1 (TBA). *For a function* tag *and a logical language* $\langle P, L, \models \rangle$*, where*

- tag*:* $S \cup O \rightarrow 2^T$ *maps a subject or object to the set of tags assigned to it*
- P*: the set of all authorization rules*
- L*: the set of queries including* allow(*s,o,r*) *for all subjects* s*, objects* o*, rights* r
- $\models$*: a subset of* $P \times \mathsf{Tag} \times L$

$\mathsf{auth}_{TBA}(s, o, r)$ *if and only if* $P', \mathsf{tag} \models \mathsf{allow}(s, o, r)$ *for some* $P' \subseteq P$.

The operator $\models$ dictates which queries are true given the set of authorization rules P and a function tag. The following example based on the scenario introduced in Section 1 illustrates TBA using Datalog as the policy language.

Example 1 (TBA). Consider two subjects s_1 and s_2 and two objects o_1 and o_2 that are tagged as follows:

- tag(s_1) = {uk_navy,operation_atalanta,operations_specialist}
- tag(s_2) = {fr_navy}
- tag(o_1) = {cargo_ship,radar}
- tag(o_2) = {gulf_of_aden,sat_732,high_res}

Further, consider the following policy rules:

1. allow(S_x,O_x,read) :- uk_navy $\in$ tag(S_x), cargo_ship $\in$ tag(O_x)
2. allow(S_x,O_x,read) :- fr_navy $\in$ tag(S_x), cargo_ship $\in$ tag(O_x)
3. allow(S_x,O_x,read) :- operations_specialist $\in$ tag(S_x), radar $\in$ tag(O_x)
4. allow(S_x,O_x,read) :- uk_navy $\in$ tag(S_x), operation_atalanta $\in$ tag(S_x), high_res $\in$ tag(O_x), sat_732 $\in$ tag(O_x)

This policy allows the members of the British and French Navy (denoted uk_navy and fr_navy respectively) to access documents about cargo ships (rules 1 and 2), operations specialists to access documents about radar systems (rule 3), and all members of the British Navy serving on Operation Atalanta to access high resolution satellite photographs taken by sat_732 (rule 4). As a result, subject s_1 can access objects o_1 and o_2, while subject s_2 can only access object o_1.

Tag-based authorization differs from standard logical access control models in that the tag function has a fixed semantics defined outside of the policy. In particular, the semantics of tag is defined by the users of a system, who assign tags to the system's subjects and objects. The fixed semantics of tag forces security administrators to define authorization policies at a higher level of abstraction than the usual $S \times O \times R$. More precisely, TBA policies are defined over the space of tags $2^T \times 2^T \times R$, i.e., subjects and objects are replaced by tag sets. This abstraction might result in a less flexible system if the tag-space is not exhaustive enough to accommodate a particular situation; however, the model can be easily adapted to define policies over $(S \cup 2^T) \times (O \cup 2^T) \times R$ (which combines the space of tags and the ones of subjects and objects) by adding to the set of tags T a tag identifying each subject and object in a system.

In [15], the authors evaluate the expressive power of TBA by expressing a range of well-known policy idioms. In particular, they show that TBA can be successfully employed to represent an access matrix, attribute-based access control policies, role-based access control policies, discretionary access control, mandatory access control, and three rule types of the RT [18] policy language (namely, all rule types except for "linked roles"). In addition, their results show that TBA is strictly more expressive than common access control models such as SDCO [21], ARBAC97 [23], and BLP [3].

2.2 Limitations of TBA

The applicability of TBA is due to a key observation: within a system, the users responsible for creating and categorizing (i.e., tagging) data are usually not in charge and do not have the expertise to define the security policies governing the system. For example, on a navy vessel some operators have the task of analyzing the surrounding maritime traffic and identifying suspicious behaviors, other operators gather intelligence and add details to these suspicious activities, etc. The policy governing the access to this information, on the other hand, is defined by the authorities in command of operations.

By assigning tags to subjects and objects, however, users directly influence the authorizations within a system. For this reason, it is necessary to enable policy-writers (e.g., security administrators) to control the tagging process. In this respect, we identify two key issues that are not addressed by the TBA model:

1. *Administrative policies:* Security administrators should be able to specify policies defining which users are allowed to assign and revoke which tags to which subjects and objects. For instance, the security administrator of a navy vessel should be able to restrict to the commanding officer the right to promote the vessel's officers to higher ranks. In addition, it should be possible to regulate the propagation of users' rights, i.e., the extent to which a user can delegate its tagging rights to other users.

2. *Tag Verification and Revocation:* It should be possible to hold users accountable for the tags they assign, and to verify the compliance of tags with administrative policies by checking who assigned them. Consider, for instance, a document containing information about a suspected pirate attack, distributed by the French Navy to its allies in Operation Atalanta. If some tag is added to the document by a user outside the navy's system, the security administrator of the French Navy should be able to verify whether the user was authorized to label the document (e.g., if the user is an operator of an allied navy, rather than an unknown subject). Accordingly, "invalid" tags identified in the verification process should be revoked.

Even though some simple administrative policies could be expressed in TBA, the fact that the "issuer" of a tag is not taken into account by the model makes it impossible to specify constraints that link tags based on the subject who assigned them (e.g., RT's linked roles [18]). Therefore, TBA does not allow the specification of rules such as "a subject can revoke only the tags that she assigned", or "a subject can mark a document as sensitive only if she is (tagged as) a senior officer by a navy that the EU labels as member of Operation Atalanta".

More than the specification of administrative policies, however, the major limitation of TBA is represented by the lack of mechanisms for verifying their enforcement. In a centralized system, administrative polices can be enforced by means of an access control system that governs the tagging process. Tag verification can be achieved by means of a logging mechanism that records all the tags assigned by the system users. Security administrators can then simply audit these logs to verify their compliance with the system's policies, and revoke the invalid tags identified in the process. In a distributed system, however, these solutions are insufficient for the following reasons:

1. They require entities in one security domain to trust the administrative policies (and their enforcement) of systems in different security domains.
2. Since tags may be assigned by users of different systems, and the issuer of a tag is not considered by the TBA model, verifying the compliance of the tags assigned by a subject with respect to administrative policies might not be feasible. In fact, this might require inspecting the logs of possibly all the systems in the distributed system. It is unlikely, however, that a system would disclose its logs to systems from a different security domain for auditing purposes. In addition, as information is exchanged between different systems, tracing the tags assigned by a certain user (e.g., to revoke them) becomes very difficult, if not impossible.

In the next section we show how the TBA model can be extended to address these limitations.

3 The TBA^2 Model

In this section we present the Tag-Based Authorization & Administration (TBA^2) model, an extension of TBA that enables policy administration in distributed systems. TBA^2 extends TBA by associating an *issuer* to each tag, which identifies the user who assigned the tag. Intuitively, tags become signed statements issued by a user within a system. Formally, we modify the definition of the tag function introduced in Section 2.1 as follows:

$\mathsf{tag} : S \cup O \rightarrow 2^{S \times T}$, which returns the set of issuer-tag pairs associated to a subject or object. Representing tags as signed statements is a first step towards addressing the limitations of TBA mentioned in Section 2.2. In the next subsections we discuss how TBA^2 can solve those limitations in details.

For the sake of simplicity, the solution we propose is based on the assumption that the set T of possible tags that a user can assign is common to all the systems in the distributed system. In a real-world distributed system, however, the set of assignable tags might vary from system to system, as entities in different security domains might employ different terms to denote similar concepts. Semantic alignment techniques [14, 24] could be required to align the systems' vocabularies. An additional assumption we make is that each subject and object *belongs* to one system, which represents the security domain where a subject operates, or the system that owns an object (or that has exclusive rights on it). Given a subject or object identifier, it is possible to determine the system to which the subject or object belongs.

3.1 Authorization and Administration Policies

Administrative policies constrain the tags that a user is authorized to assign or revoke. The advantage of TBA^2 with respect to TBA is that it links every tag to the user who assigned it, enabling security administrators to specify fine-grained administrative policies.

TBA^2 requires the set of rights R to include the rights $assign\_tag$ and $revoke\_tag$. Then, TBA^2 is defined as follows.

Definition 2 (TBA^2). *For a function* tag *and a logical language* $\langle P \cup A, L, \models \rangle$*, where*

- tag*:* $S \cup O \rightarrow 2^{S \times T}$ *returns the issuer-tag pairs associated to a subject or object*
- P*: the set of all authorization rules*
- A*: the set of all administrative rules*
- L*: the set of queries including* $\mathsf{allow}(s, o, r)$*,* $\mathsf{allow}(s, o, r', ST)$*, and* $\mathsf{allow}(s, s', r', ST)$ *for all subjects* s *and* s'*, objects* o*, rights* r*, right* $r' \in \{assign\_tag, revoke\_tag\}$*, and set of signed tags* $ST \subseteq 2^{S \times T}$
- $\models$*: a subset of* $(P \cup A) \times \mathsf{Tag} \times L$

  $\mathsf{auth}_{TBA^2}(s, o, r)$ *if and only if* $P', \mathsf{tag} \models \mathsf{allow}(s, o, r)$
  $\mathsf{auth}_{TBA^2}(s, o, r', ST)$ *if and only if* $A', \mathsf{tag} \models \mathsf{allow}(s, o, r', ST)$
  $\mathsf{auth}_{TBA^2}(s, s', r', ST)$ *if and only if* $A', \mathsf{tag} \models \mathsf{allow}(s, s', r', ST)$

for some $P' \subseteq P$*,* $A' \subseteq A$*.*

The following example presents TBA^2 authorization and administrative policies.

Example 2 (TBA^2). Consider three subjects s_1, s_2, and s_3 and an object o belonging to a system governed by the British Navy, which are tagged as follows:

- $\mathsf{tag}(s_1)$ = {(uk_navy,senior_officer)}
- $\mathsf{tag}(s_2)$ = {(uk_navy,junior_officer)}
- $\mathsf{tag}(s_3)$ = {(uk_navy,junior_officer)}
- $\mathsf{tag}(o)$ = {(uk_navy,secret)}

The subjects' tags are issued by the British Navy (denoted as "uk_navy") and indicate that subject s_1 has rank senior_officer, while s_2 and s_3 have rank junior_officer. Object o is tagged by the British Navy as secret. The access to object o and the administration of rights within the system are regulated by the following rules:

1. allow(S_x,o,read) :- (eu,navy) $\in$ tag(S_y), (S_y,senior_officer) $\in$ tag(S_x)
2. allow(S_x,O_x,$assign\_tag$,{(S_x,T_x)}) :- (eu,navy) $\in$ tag(S_y),
   (S_y,senior_officer) $\in$ tag(S_x),
   (eu,navy) $\in$ tag(S_z), (S_z,secret) $\in$ tag(O_x)
3. allow(S_x,S_y,$assign\_tag$,{(S_x,senior_officer)}) :- (eu,navy) $\in$ tag(S_z),
   (S_z,senior_officer) $\in$ tag(S_x),
   (S_z,junior_officer) $\in$ tag(S_y)
4. allow(S_x,O_x,$revoke\_tag$,{(S_x,T_x)}) :- (S_x,T_x) $\in$ tag(O_x)
5. allow(S_x,S_y,$revoke\_tag$,{(S_x,T_x)}) :- (S_x,T_x) $\in$ tag(S_y)

The first rule is an authorization rule stating that object o can be read by a subject S_x if S_x is labeled as senior officer by an EU navy S_y. Rules 2, 3, 4 and 5 are administrative rules. Rule 2 allows senior officers of EU navies to assign tags to objects labeled as secret by any EU navy. Rule 3 allows senior officers of EU navies to assign a *senior_officer* tag to junior officers of the same navy (therefore delegating their rights). Finally, rules 4 and 5 allow the issuer of a tag to revoke the tag. Assuming tag (eu,navy) $\in$ tag(uk_navy), the policy allows subject s_1 to read object o and to assign a *senior_officer* tag to subjects s_2 and s_3.

3.2 Semantics of Administrative Policies

The effects of the exercise by a subject s of the administrative rights $assign\_tag$ and $revoke\_tag$ for a set of (signed) tags ST are shown in Figure 1. The effects of invoking allow(s, o, $assign\_tag$, ST) are intuitive, and imply that a tag st (for each $st \in ST$) is added to the set tag(o). On the other hand, the assignment of a set of tags ST by a subject s to a subject s' can be seen as the delegation of some of the rights (or roles) of s to s'. Delegation can be implemented according to two models: *grant* or *transfer* [8]. In the grant model, after a successful delegation both s and s' are able to benefit from the delegated rights or roles. On the contrary, according to the transfer model subject s loses the delegated rights or roles. Figure 1(a) shows the effects of invoking allow(s, s', $assign\_tag$, ST) using the grant model. The transfer model would imply that all the tags in ST are removed from the set tag(s) after being added to tag(s').

Similarly to the tag assignment operation, also the effects of invoking allow(s, s', $revoke\_tag$, ST) depend on the revocation model employed by the system. To motivate the existence of different revocation models, we describe a scenario based on the tags and rules in Example 2. Assume that senior officer s_1 wants to temporarily delegate her rights to junior officer s_2 because of an emergency. Then, s_1 assigns a *senior_officer* tag to s_2. Later, subject s_2 delegates her rights to s_3, and accordingly assigns tag *senior_officer* to s_3. When the emergency is over, s_1 returns to her regular duties and revokes the *senior_officer* tag from s_2. Now, the question is whether s_3's *senior_officer* tag should also be automatically revoked or not.

allow(s,o,assign_tag,ST)
$\forall\ st \in ST$: $\mathsf{tag}(o) = \mathsf{tag}(o) \cup \{st\}$

(a) Semantics of $\mathsf{allow}(s, o, assign\_tag, ST)$ and $\mathsf{allow}(s, s', assign\_tag, ST)$

allow(s,s′,assign_tag,ST)
$\forall\ st \in ST$:
$\mathsf{tag}(s') = \mathsf{tag}(s') \cup \{st\}$

allow(s,s′,revoke_tag,ST)
$\forall\ st \in ST$: $\mathsf{tag}(s') = \mathsf{tag}(s') \backslash \{st\}$

let $ST = \{(s_1, t_1), \ldots, (s_n, t_n)\}$
$\forall\ so \in S \cup O, t \in T$ such that $(s', t) \in \mathsf{tag}(so)$
if $\exists A' \subseteq A$ such that
$A', \mathsf{tag} \models \mathsf{allow}(s', so, assign\_tag, \{(s', t)\})$ if $(s_1, t_1), \ldots, (s_n, t_n) \in \mathsf{tag}(s')$
and $\forall A'' \subseteq A$ we have that
$A'', \mathsf{tag} \not\models \mathsf{allow}(s', so, assign\_tag, \{(s', t)\})$ if $(s_1, t_1), \ldots, (s_n, t_n) \notin \mathsf{tag}(s')$
then
if $so \in S$ then
let ST' be the set of such tags (s', t)
invoke $\mathsf{allow}(s', so, revoke\_tag, ST')$
else
$\mathsf{tag}(so) = \mathsf{tag}(so) \backslash \{(s', t)\}$

(b) Semantics of $\mathsf{allow}(s, s', revoke\_tag, ST)$ with Cascading Revocation

allow(s,s′,revoke_tag,ST)
$\forall\ st \in ST$: $\mathsf{tag}(s') = \mathsf{tag}(s') \backslash \{st\}$
let $ST = \{(s_1, t_1), \ldots, (s_n, t_n)\}$ $\forall\ so \in S \cup O, t \in T$ such that $(s', t) \in \mathsf{tag}(so)$ if $\exists A' \subseteq A$ such that $A', \mathsf{tag} \models \mathsf{allow}(s', so, assign\_tag, \{(s', t)\})$ if $(s_1, t_1), \ldots, (s_n, t_n) \in \mathsf{tag}(s')$ and $\forall A'' \subseteq A$ we have that $A'', \mathsf{tag} \not\models \mathsf{allow}(s', so, assign\_tag, \{(s', t)\})$ if $(s_1, t_1), \ldots, (s_n, t_n) \notin \mathsf{tag}(s')$ then $\mathsf{tag}(so) = \mathsf{tag}(so) \backslash \{(s', t)\} \cup \{(s, t)\}$

(c) Semantics of $\mathsf{allow}(s, s', revoke\_tag, ST)$ with Non-Cascade Revocation

allow(s,o,revoke_tag,ST)
$\forall\ st \in ST$: $\mathsf{tag}(o) = \mathsf{tag}(o) \backslash \{st\}$

(d) Semantics of $\mathsf{allow}(s, o, revoke\_tag, ST)$

Fig. 1. Effects of the invocation of rights $assign\_tag$ and $revoke\_tag$

Two main revocation models have been proposed in the literature. The first model, called *cascading* revocation [4, 12], aims to overturn all the changes to a system authorizations made exploiting the tags being revoked. This implies that if a subject s revokes a set of tags ST from a subject s', then all tags subsequently assigned by s' (and by the subjects to which s' assigned a tag) without other supporting authorizations must be recursively revoked. The effects of invoking $\mathsf{allow}(s, s', revoke\_tag, ST)$ with cascading revocation are shown in Figure 1(b). A domain that typically resorts to cascading revocation is data protection. Whenever an individual revokes the consent (i.e., the right) to process her personal data to a service provider, all the authorizations on the data of the

service provider and of the subcontractors to whom the service provider delegated the processing of the data are revoked.

The dual model of cascading revocation is called *non-cascade* revocation [6] (or *simple* revocation in [4]). In non-cascade revocation, if a subject s revokes a set of tags ST from a subject s', instead of revoking the tags that s' assigned exploiting the authorizations deriving from ST (as done by cascading revocation), these tags are modified as if they were issued by s. Intuitively, this requires s to be allowed to both revoke tags ST from subject s' and to assign tags ST in her place. The rationale behind non-cascade revocation is clarified by the following example. In most organizations, the authorizations that users possess are related to their role within the organization. Suppose there is a change in the role of a user s'. This may imply a change also in the privileges of s': new rights will be granted to s' and some of her previous rights will be revoked. Applying cascading revocation would result in the undesirable effect of deleting all the authorizations that s' granted and, recursively, all the authorizations granted through them, which then might need to be re-issued. Moreover, all the tags assigned by s' that depend on the revoked rights would be invalidated. A better solution to this scenario is to preserve the authorizations granted by user s', possibly substituting s' with another user as the grantor (i.e., issuer) of those authorizations. In [6], for instance, s' is replaced by the user s who is revoking her rights. The semantics of $\mathsf{allow}(s, s', revoke\_tag, ST)$ with non-cascade revocation is shown in Figure 1(c).

According to both semantics presented above, since objects cannot further delegate their rights to other subjects or objects, the effects of invoking $\mathsf{allow}(s, o, revoke\_tag, ST)$ are the same in cascading and non-cascade revocation (Figure 1(d)).

3.3 Tag Verification

Tag verification is the process of verifying the compliance of tags with administrative policies. More precisely, the goal of tag verification is to determine whether a user is (or was) authorized to assign a given tag. Typically, the enforcement of authorization and administrative policies within a system is achieved by means of a priori access control mechanisms. In a distributed system, however, relying exclusively on a priori mechanisms requires entities in one security domain to trust systems in different security domains for policy enforcement. Thanks to the observability of users' actions deriving from the signing of tags, TBA^2 allows security administrators to complement a priori mechanisms with a posteriori tag verification using a lightweight auditing mechanism.

Technically, we say that a tag t assigned by a subject s to an object o (respectively to a subject s') is *valid* if for a set of administrative rules A' and a set of signed tags ST such that $(s, t) \in ST$ we have that

$$A', \mathsf{tag} \models \mathsf{allow}(s, o, assign\_tag, ST) \text{ (resp. } A', \mathsf{tag} \models \mathsf{allow}(s, s', assign\_tag, ST))$$

Otherwise, we say that the tag is *invalid*. This validity check can be used both as an a priori mechanism for the enforcement of administrative policies and a posteriori for auditing purposes.

The verification of a tag (s, t) against the administrative policy of a system might require the verification of a set of tags $(s_1, t_1), \ldots, (s_n, t_n)$ against the policies of systems

in different security domains. In fact, to verify whether tag (s, t) is valid, we need in turn to verify the validity of the *supporting tags* of (s, t), i.e., the tags that authorized subject s to issue (s, t). Consider, for instance, rule 3 in Example 2, which states that "senior officers of EU navies may assign a $senior\_officer$ tag to junior officers of the same navy". To verify whether the $senior\_officer$ tag assigned by subject s_1 to subject s_2 is valid, we need first to confirm that s_1 is actually a senior officer and s_2 has a $junior\_officer$ tag issued by the same navy. This verification process is similar to the credential chain discovery problem in trust management. Accordingly, trust management algorithms [19, 25] can be employed to support the verification of tags' validity.

An additional problem of tag verification is that in a distributed system entities in one security domain might not trust systems in different security domain to perform the validity check. In this respect, we identify three types of trust relationships that can be of interest for tag verification in TBA^2, resulting in three possible verification strategies. In what follows, we refer to the object (resp. subject) to which a tag is assigned as the *target* object (resp. subject) of the tag. The first possible verification strategy is *issuer verification* of tags, where the system to which the issuer of a tag belongs is trusted for checking the validity of the tag. The second strategy is *target verification*, which enables systems to verify the validity of the tags assigned to an object according to the intention of the object's owner. A possible application scenario for target verification is the protection of digital media, where only the content owner is entitled to define the authorizations to access the object. Finally, *local verification* of tags can be employed in scenarios where there is no mutual trust among systems in different security domains. With local verification, the system which is interested in verifying the validity of a tag performs the check with respect to its local administrative policy.

The following example illustrates local verification of tags based on the administrative policy in Example 2.

Example 3 (Tag Verification). The French Navy distributes a document d containing the location of a suspected pirate attack to the other navies involved in Operation Atalanta. When d is received by the British Navy, it contains the following tags:

- $\mathsf{tag}(d)$ = {(fr_navy,secret),(s_4,inaccurate_information)}

where subject s_4 is labeled as follows:

- $\mathsf{tag}(s_4)$ = {(it_navy,reconnaissance_pilot)}

Since in Example 2 there is no rule defined by the British Navy that implies $\mathsf{allow}(s_4, d, assign\_tag, \{(s_4, inaccurate\_information)\})$, the tag added by subject s_4 is considered invalid. In fact, the policy of the British Navy allows only senior officers to tag documents marked as secret by an EU navy. The British Navy might thus decide to proceed with further investigations before concluding the inaccuracy of d's information.

For the sake of simplicity, the auditing mechanism discussed in this section verifies the compliance of tags with respect to the administrative policies that are currently in force. In other words, a tag is considered valid if its assignment is authorized by the administrative policy in force at the moment in which the tag is *verified*. An alternative verification mechanism could verify tags with respect to the administrative policy in force at the moment in which the tag was *assigned*. Intuitively, the implementation of

the latter mechanism is more complex and requires, e.g., timestamped tags and repositories containing all the administrative policies adopted by a system over time. An extension of the TBA^2 model in this direction is discussed in Section 6.

3.4 Tag Revocation

Whenever security administrators identify invalid tags, they should revoke them to preserve the consistency of function tag with respect to administrative policies. In a centralized system, revocation can be performed by simply deleting incorrect tags from the system. In a distributed system, revoking a set of tags is more complicated because it might not be possible to trace the tags issued by a given subject, and security administrators cannot delete tags assigned to subjects and objects residing in different security domains. TBA^2 enables a simple solution to this problem, where security administrators communicate the issuer-tag pairs to be revoked to other systems by broadcasting or publishing in an appropriate location *revocation lists* of tags. The recipient systems can then decide whether to revoke the listed tags or ignore the recommendation.

Definition 3 (Revocation List). *A revocation list is a triple $\langle s, so, T' \rangle$ where $T' \subseteq T$ is a set of tags, s is the issuer of the tags in T', and so is the target subject or object.*

Intuitively, a revocation list contains the set T' of tags assigned by a subject s to subject or object so which should be revoked according to the system publishing the revocation list.

Example 4 (Revocation List). The revocation list for the invalid tag identified by the British Navy in Example 3 is the following: $\langle s_4, d, \{inaccurate\_information\} \rangle$. The revocation list is published by the British Navy on its public record of invalid tags.

The decision of the security administrator of a system sys on whether to actually revoke the set of tags listed in a revocation list $\langle s, so, T' \rangle$ published by a system sys' is strictly correlated to the verification strategy employed by sys. If sys resorts to issuer (resp. target) verification, for instance, sys trusts the system to which s (resp. so) belongs to perform the validity check of tags T'. Consequently, if s (resp. so) belongs to sys', the security administrator of sys is likely to delete tags T' from its system. On the contrary, if sys resorts to local verification, it might decide to proceed with further investigations before deleting the tags. The revocation list published by the British Navy in Example 4, for instance, might be taken into consideration by the vessels of the British Navy, but ignored by the vessels of other EU countries, because derived through local verification with respect to the British Navy's policy.

As an alternative to revocation lists, we consider the use of *negative tags*. Negative tags are signed tags that state that a certain tag assigned to a subject or object is not (or no longer) valid according to the issuer of the negative tag. The advantage of negative tags is that they enable the verification of revoked tags, which might not be possible if the tags are deleted. However, this might lead to a very large number of tags (both "positive" and negative) assigned to each subject and object.

Rather than simply deleting invalid tags (or issuing negative tags), other approaches can be employed for restoring the compliance of tags with administrative policies.

In some critical or uncertain situation, for instance, security administrators might decide to simply highlight the invalid tags and refer to a competent user for determining what to do with them. Alternatively, systems may rely on *repair constraints* [11] to determine how to handle invalid tags. A repair constraint might, for example, dictate in which conditions cascading rather than non-cascade revocation should be applied on a tag.

4 Evaluation of TBA^2

In this section we evaluate the expressive power of the TBA^2 model. First, it is easy to demonstrate that TBA^2 is strictly more expressive than TBA. In fact, by not bounding the tags' issuers, TBA^2 can express exactly the same constraints definable by TBA. On top of this, associating an issuer to each tag enables the specification of authorization and administrative rules discriminating based on the issuer of tags, such as for instance linked roles in RT [18]. A linked role is a rule of the form $A.r \leftarrow B.r_1.r_2$, which states that subject A assigns a subject S_x (implicitly defined) to role r if S_x is labeled as a member of role r_2 by a subject S_y who is assigned to role r_1 by subject B. The reason why linked roles cannot be represented in TBA is that they require the binding of the subject of the first role r_1 to the issuer of the second role r_2. TBA^2 rules 1, 2, and 3 in Example 2 are examples of RT's linked roles.

We now show how TBA^2 can represent policies from two reference administrative models proposed in the literature, namely the Harrison, Ruzzo, and Ullman (HRU) model [13] and ARBAC97 [23]. The HRU model [13] employs an access matrix for the specification of the rights of users on the objects in a system, and relies on a set of commands for modifying users' authorizations. The model includes three predefined commands: commands CONFER and REVOKE allow the owner of an object to respectively grant to and revoke from other subjects any right on the objects she owns; command TRANSFER allows users to delegate their rights to other users. In TBA^2 we use $\mathsf{allow}(s, o, assign\_tag, T')$ to define commands CONFER and TRANSFER, and $\mathsf{allow}(s, o, revoke\_tag, T')$ to define the REVOKE command. We assume the tags in T to consist of pairs $\langle s, r\rangle$, representing each possible right $r \in R$ of a subject $s \in S$. The tags associated to an object define the rights of the users of a system on that object.

Example 5 (Mapping HRU to TBA^2). The following three rules define commands *CONFER*, *TRANSFER*, and *REVOKE* respectively:

1. $\mathsf{allow}(\mathrm{S}_x,\mathrm{O}_x,assign\_tag,\{(\mathrm{S}_y,\mathrm{R}_x)\})$:- $(sys,\langle \mathrm{S}_x,\mathrm{own}\rangle) \in \mathsf{tag}(O_x)$
2. $\mathsf{allow}(\mathrm{S}_x,\mathrm{O}_x,assign\_tag,\{(\mathrm{S}_y,\mathrm{R}_x)\})$:- $(sys,\langle \mathrm{S}_z,\mathrm{own}\rangle) \in \mathsf{tag}(O_x)$, $(\mathrm{S}_z,\langle \mathrm{S}_x,\mathrm{R}_x^*\rangle) \in \mathsf{tag}(O_x)$
3. $\mathsf{allow}(\mathrm{S}_x,\mathrm{O}_x,revoke\_tag,\{(\mathrm{S}_y,\mathrm{R}_x)\})$:- $(sys,\langle \mathrm{S}_x,\mathrm{own}\rangle) \in \mathsf{tag}(O_x)$

The first rule states that the owner S_x of an object O_x can assign any right R_x on O_x to any subject S_y. The tag representing the ownership of an object is a "system tag"; we use sys to denote the issuer of system tags. Rule 2 represents the right of a subject S_x to delegate a right R_x on object O_x to a subject S_y, provided that R_x is a transferable right (denoted by symbol * in the HRU model) assigned to S_x by the owner S_z of object O_x. Finally, rule 3 states that the owner of an object can revoke any right on that object.

Note that the HRU model allows users to define additional commands to modify the authorizations within a system. Since those commands are arbitrary, and are not described in the model, we cannot evaluate the expressiveness of TBA^2 with respect to them.

Next, we show how to represent in TBA^2 the administrative policies supported by ARBAC97 [23], an administrative model for role-based access control. In ARBAC97, roles are divided into two classes: *administrative roles* and *regular roles*. Both classes of roles are organized into hierarchies, where each role inherits all the rights assigned to the children nodes in the hierarchy. The ARBAC97 model relies on four commands for the specification of administrative policies:

1. $can\_assign(ar, \phi, \{rr_1, \ldots, rr_n\})$
2. $can\_revoke(ar, \{rr_1, \ldots, rr_n\})$
3. $can\_assignp(ar, \phi, \{rr_1, \ldots, rr_n\})$
4. $can\_revokep(ar, \{rr_1, \ldots, rr_n\})$

where ϕ (called prerequisite condition) is a boolean expression on regular roles, which defines the requirements on the membership (or non-membership) of a user to some roles. Commands (1) and (2) are used to specify the right to assign and revoke roles, while commands (3) and (4) define the rights to assign and revoke permissions. More precisely, command (1) defines the right of a member of the administrative role ar (or a member of an administrative role above ar in the hierarchy) to assign to a user who satisfies the prerequisite conditions ϕ the membership to regular roles $rr_1, \ldots, rr_n$. Command (2) assigns to members of the administrative role ar (or higher roles in the hierarchy) the right to revoke regular roles $rr_1, \ldots, rr_n$. Similarly, command (3) allows members of the administrative role ar (or higher) to assign to roles $rr_1, \ldots, rr_n$ any permission whose assignment to regular roles satisfies ϕ, and command (4) enables members of ar (or higher) to revoke any right to roles $rr_1, \ldots, rr_n$. To represent ARBAC97, we consider a set of administrative roles AR and regular roles RR to be defined as tags in T. The assignment of a user to a role is represented by the assignment of a tag from RR to the user. In addition, similarly to the previous example, we employ tags consisting of pairs $\langle s, r \rangle$ to represent a right $r \in R$ of a subject $s \in S$. Finally, for representing a prerequisite condition ϕ, we rewrite ϕ in disjunctive normal form, i.e., into a formula of the form $(CR_{11} \wedge \ldots \wedge CR_{1m_1}) \vee \ldots \vee (CR_{p1} \wedge \ldots \wedge CR_{pm_p})$, where CR_{ij} (with $i \in \{1, \ldots, p\}$, $j \in \{1, \ldots, m_i\}$) is either cr_{ij} or $\neg cr_{ij}$, with $cr_{ij} \in RR$, and the negation symbol $\neg$ denotes non-membership to a regular role. Negation as failure is employed to interpret negated roles: a user is not a member of a role cr_{ij} if she is not assigned a tag (s, cr_{ij}), for any $s \in S$.

Example 6 (Mapping ARBAC97 to TBA^2). The following TBA^2 rules define ARBAC97 commands (1), (2), (3), and (4) respectively:

1. allow(S_x,S_y,*assign_tag*,{(S_x,rr_1),. . .,(S_x,rr_n)}) :- (∗,ar) ∈ tag(S_x), (∗,cr_{11}) ⊙ tag(S_y),
   . . ., (∗,cr_{1m_1}) ⊙ tag(S_y)
   . . .
   allow(S_x,S_y,*assign_tag*,{(S_x,rr_1),. . .,(S_x,rr_n)}) :- (∗,ar) ∈ tag(S_x), (∗,cr_{p1}) ⊙ tag(S_y),
   . . ., (∗,cr_{pm_p}) ⊙ tag(S_y)

2. $\mathsf{allow}(\mathrm{S}_x,\mathrm{S}_y,revoke\_tag,\{(*,\mathrm{rr}_1),\ldots,(*,rr_n)\})$:- $(*,\mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x)$
3. $\mathsf{allow}(\mathrm{S}_x,\mathrm{S}_y,assign\_tag,\{(\mathrm{S}_x,\langle \mathrm{S}_y,\mathrm{R}_x\rangle)\})$:- $(*,\mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x)$, $(*,rr_1) \in \mathsf{tag}(\mathrm{S}_y)$, $(*,cr_{11}) \in \mathsf{tag}(S_{cr_{11}})$, $(*,\langle \mathrm{S}_{cr_{11}},\mathrm{R}_x\rangle) \odot \mathsf{tag}(O_{cr_{11}})$, $\ldots$, $(*,cr_{1m_1}) \in \mathsf{tag}(S_{cr_{1m_1}})$, $(*,\langle \mathrm{S}_{cr_{1m_1}},\mathrm{R}_x\rangle) \odot \mathsf{tag}(O_{cr_{1m_1}})$
   $\ldots$
   $\mathsf{allow}(\mathrm{S}_x,\mathrm{S}_y,assign\_tag,\{(\mathrm{S}_x,\langle \mathrm{S}_y,\mathrm{R}_x\rangle)\})$:- $(*,\mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x)$, $(*,rr_n) \in \mathsf{tag}(\mathrm{S}_y)$, $(*,cr_{p1}) \in \mathsf{tag}(S_{cr_{p1}})$, $(*,\langle \mathrm{S}_{cr_{p1}},\mathrm{R}_x\rangle) \odot \mathsf{tag}(O_{cr_{p1}})$, $\ldots$, $(*,cr_{pm_p}) \in \mathsf{tag}(S_{cr_{pm_p}})$, $(*,\langle \mathrm{S}_{cr_{pm_p}},\mathrm{R}_x\rangle) \odot \mathsf{tag}(O_{cr_{pm_p}})$
4. $\mathsf{allow}(\mathrm{S}_x,\mathrm{S}_y,revoke\_tag,\{(*,\langle \mathrm{S}_y,\mathrm{R}_x\rangle)\})$:- $(*,\mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x)$, $(*,rr_1) \in \mathsf{tag}(\mathrm{S}_y)$
   $\ldots$
   $\mathsf{allow}(\mathrm{S}_x,\mathrm{S}_y,revoke\_tag,\{(*,\langle \mathrm{S}_y,\mathrm{R}_x\rangle)\})$:- $(*,\mathrm{ar}) \in \mathsf{tag}(\mathrm{S}_x)$, $(*,rr_n) \in \mathsf{tag}(\mathrm{S}_y)$

where $*$ indicates any subject in S, and $\odot$ is either $\in$ or $\notin$ depending on the corresponding element in ϕ. The first set of rules allows a subject S_x with administrative role ar to assign to a subject S_y whose roles satisfy the formula $(CR_{i1} \wedge \ldots \wedge CR_{im_i})$ (for any $i \in \{1,\ldots,p\}$) to regular roles $rr_1,\ldots,rr_n$. The second rule allows a member S_x of administrative role ar to revoke to a subject S_y regular roles $rr_1,\ldots,rr_n$, independently from the subject who assigned them. The set of rules in item 3 states that a subject S_x who is a member of administrative role ar may assign a right R_x to a subject S_y, provided that S_y is a member of regular role rr_j (with $j \in \{1,\ldots,n\}$), and R_x is a right whose assignment to regular roles satisfies $(CR_{i1} \wedge \ldots \wedge CR_{im_i})$. Finally, the set of rules in item 4 gives to a subject S_x having administrative role ar the right to revoke any right to the members of role rr_j.

In the example above we do not consider inheritance of rights among roles in a hierarchy. Rather, we assume that a rule is defined for each administrative role ar having a certain right. A role hierarchy could be easily defined using a predicate $higher\_role(ar_1, ar_2)$, and adding to each rule a condition $higher\_role(ar, ar_{min})$, where ar_{min} is the minimum role in the hierarchy to which the rule applies. In addition, we slightly modify the semantics of commands (3) and (4). Whereas in ARBAC97 permissions are assigned to roles, in our representation they are assigned to the members of a role. From the practical point of view, however, the two semantics are equivalent.

The examples above demonstrate that TBA^2 can express the administrative constraints defined by HRU and ARBAC97. As a matter of fact, neither HRU nor ARBAC97 fully exploit the expressiveness of TBA^2. As shown by Example 6, for instance, ARBAC97 does not exploit the capability of TBA^2 of constraining the issuer of a tag and the rights that a member of an administrative role may assign. With respect to the HRU model, TBA^2 allows for the specification of much more complex constraints than those defined in commands *CONFER*, *TRANSFER*, and *REVOKE*, e.g., based on the properties of subjects and objects. This implies that, in terms of expressive power, TBA^2 represents a more comprehensive solution than the considered models.

5 Related Work

TBA has been studied informally in [20, 26], though that work allows tags on subjects but not on objects. Next to it, substantial work has been done on logical access control models, both based on Datalog (e.g., [2, 18, 22]) as well as on more expressive logics (e.g., [1, 9, 27]). While many of the existing logical access control languages can be used to encode tag-based authorization policies, it is the commitment to document and user tagging (an activity that can be carried out by users with a wide range of technical expertise) that makes TBA useful to a broad class of organizations.

The work related to the contributions of this paper spans two main topics: policy administration and auditing mechanisms. While many access control models for distributed systems have been proposed in the literature, policy administration received much less consideration. A number of administration models exist [4, 5, 10, 13, 17, 23], but they focus mainly on the expressivity of administrative policies, and do not consider the challenges associated with their enforcement in a distributed setting. The innovation of TBA^2 in this respect lies in the fact that it allows for an easy verification of policy compliance, thus not requiring entities in one security domain to trust systems in different security domains for the enforcement of administrative policy. In addition, we have shown that TBA^2 is more expressive than two reference administrative models, namely ARBAC97 [23] and HRU [13].

Similarly to TBA^2, the existing a posteriori solutions (e.g., [7]) perform the verification of policy compliance through auditing mechanisms. However, to achieve this, they rely on logging mechanisms that record users' actions, and trusted auditing authorities that verify the compliance of those actions with policies. Our model represents a lightweight solution for policy compliance verification that does not require the realization of such an auditing infrastructure. We propose the use of trust management algorithms [19, 25] to support the verification of policy compliance.

6 Discussion and Conclusions

In this paper we have introduced TBA^2, an extension of the TBA model [15] that enables policy administration in distributed systems. Similarly to TBA, TBA^2 allows relatively untrained users to assign descriptive tags to a system's subjects and objects; trained security experts then write logic-based authorization policies that define access rights in terms of those tags. In addition, by linking each tag to its issuer (i.e., the user who assigned it), TBA^2 enables the specification of fine-grained administrative policies whose enforcement can be verified through a lightweight auditing technique. We have shown that our model is more expressive than TBA and than the HRU [13] and ARBAC97 [23] administrative models. Thus, TBA^2 represents a flexible, easy to use, yet expressive access control solution which matches the needs of real-world organizations.

The auditing mechanism proposed in Section 3.3 verifies tags' validity with respect to the administrative policy currently in force within a system. In some situations, however, it is preferable to verify the validity of a tag with respect to the administrative policies effective when the tag was issued. For example, assume that the commanding officer of a British Navy vessel is summoned by the EU for a meeting at the Operation

Atalanta's headquarter. Then, the commanding officer would have to temporarily delegate the command of the vessel and the deriving responsibilities and authorizations to another officer until her return. During this period, the appointed officer will have to take several decisions which might lead to the granting and revocation of authorizations to the vessel's operators and to the tagging of several data objects exchanged among the collaborating navies. With the verification mechanism presented in Section 3.3, the revocation of the officer's rights by the commanding officer upon her return would have the undesirable effect of invalidating all the authorizations and tags assigned by the officer during her command. The design of an auditing mechanism verifying tags' validity with respect to the administrative policy in force when a tag was assigned would require two main extensions to the TBA^2 model. First, it would require the association of a timestamp to each tag to demonstrate when it was issued. Second, all the administrative policies employed by a system during its lifetime would need to be stored in a repository, together with the time interval in which they were effective. Then, whenever a tag needs to be verified, its timestamp can be used to retrieve from the repository the policy that was in force when the tag was issued, against which the validity check must be performed. The resulting enforcement mechanism is similar to those used for the enforcement of history-based access control policies [16].

To conclude, we point out that the model proposed in this paper enables security administrators to verify the compliance of users' actions with respect to the administrative policies in force within a system, but provides no guarantee that these policies are correctly specified. The verification of administrative policies with respect to the desired security properties of a system can be achieved through model checking techniques [28]. Finally, we argue that even though TBA^2 is presented as an access control solution for distributed systems, also centralized systems would benefit from employing the model. In fact, the association of each tag to its issuer enhances the "observability" of user's actions, simplifying the detection of policy violations, and may be used as a discriminant by other users in the system to determine whether a certain tag should be considered valid. Signed tags are currently employed by several existing web applications and social networks (e.g., Facebook).

Acknowledgments. This work has been done in the context of the THeCS project, which is supported by the Dutch national program COMMIT. Adam J. Lee was supported in part by the US National Science Foundation under awards CNS-0964295 and CNS-1228697.

References

1. Abadi, M., Burrows, M., Lampson, B.: A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems 15(4), 706–734 (1993)
2. Becker, M.Y., Fournet, C.Y., Gordon, A.D.: SecPAL: Design and semantics of a decentralized authorization language. Journal of Computer Security 18(4), 619–665 (2010)
3. Bell, D.E.: Looking Back at the Bell-La Padula Model. In: Proceedings of ACSAC 2005, pp. 337–351. IEEE Computer Society (2005)

4. Ben-Ghorbel-Talbi, M., Cuppens, F., Cuppens-Boulahia, N., Bouhoula, A.: Revocation Schemes for Delegation Licences. In: Chen, L., Ryan, M.D., Wang, G. (eds.) ICICS 2008. LNCS, vol. 5308, pp. 190–205. Springer, Heidelberg (2008)
5. Ben-Ghorbel-Talbi, M., Cuppens, F., Cuppens-Boulahia, N., Bouhoula, A.: A delegation model for extended RBAC. Int. J. Inf. Sec. 9(3), 209–236 (2010)
6. Bertino, E., Samarati, P., Jajodia, S.: An Extended Authorization Model for Relational Databases. IEEE Trans. Knowl. Data Eng. 9(1), 85–101 (1997)
7. Cederquist, J.G., Corin, R., Dekker, M.A.C., Etalle, S., den Hartog, J.I., Lenzini, G.: Audit-based compliance control. Int. J. Inf. Sec. 6(2), 133–151 (2007)
8. Crampton, J., Khambhammettu, H.: Delegation in role-based access control. Int. J. Inf. Sec. 7(2), 123–136 (2008)
9. Crampton, J., Loizou, G., Oshea, G.: A logic of access control. The Computer Journal 44(1), 137–149 (2001)
10. Dekker, M., Crampton, J., Etalle, S.: RBAC administration in distributed systems. In: Proceedings of SACMAT 2008, pp. 93–102. ACM (2008)
11. Greco, G., Greco, S., Zumpano, E.: A logical framework for querying and repairing inconsistent databases. IEEE Trans. Knowl. Data Eng. 15(6), 1389–1408 (2003)
12. Griffiths, P.P., Wade, B.W.: An authorization mechanism for a relational database system. ACM Trans. Database Syst. 1(3), 242–255 (1976)
13. Harrison, M.A., Ruzzo, W.L., Ullman, J.D.: Protection in operating systems. Communications of the ACM 19(8), 461–471 (1976)
14. Heeps, S., Sventek, J., Dulay, N., Schaeffer Filho, A.E., Lupu, E., Sloman, M., Strowes, S.: Dynamic Ontology Mapping for Interacting Autonomous Systems. In: Hutchison, D., Katz, R.H. (eds.) IWSOS 2007. LNCS, vol. 4725, pp. 255–263. Springer, Heidelberg (2007)
15. Hinrichs, T.L., Garrison III, W.C., Lee, A.J., Saunders, S., Mitchell, J.C.: TBA: A Hybrid of Logic and Extensional Access Control Systems. In: Barthe, G., Datta, A., Etalle, S. (eds.) FAST 2011. LNCS, vol. 7140, pp. 198–213. Springer, Heidelberg (2012)
16. Koshutanski, H., Martinelli, F., Mori, P., Vaccarelli, A.: Fine-grained and History-based Access Control with Trust Management for Autonomic Grid Services. In: Proceedings of ICAS 2006, pp. 34–43. IEEE Computer Society (2006)
17. Li, N., Mao, Z.: Administration in role-based access control. In: Proceedings of ASIACCS 2007, pp. 127–138. ACM (2007)
18. Li, N., Mitchell, J.C., Winsborough, W.H.: Design of a Role-Based Trust-Management Framework. In: Proceedings of S&P 2002, pp. 114–130. IEEE Computer Society (2002)
19. Li, N., Winsborough, W.H., Mitchell, J.C.: Distributed credential chain discovery in trust management. Journal of Computer Security 11(1), 35–86 (2003)
20. Najafian Razavi, M., Iverson, L.: Supporting selective information sharing with people-tagging. In: Proceedings of CHI 2008, pp. 3423–3428. ACM (2008)
21. Osborn, S., Sandhu, R., Munawer, Q.: Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Trans. Inf. Syst. Secur. 3(2), 85–106 (2000)
22. Ribeiro, C., Zuquete, A., Ferreira, P., Guedes, P.: SPL: An access control language for security policies with complex constraints. In: Proceedings of NDSS 2011 (2001)
23. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based administration of roles. ACM Trans. Inf. Syst. Secur. 2(1), 105–135 (1999)
24. Trivellato, D., Spiessens, F., Zannone, N., Etalle, S.: Reputation-Based Ontology Alignment for Autonomy and Interoperability in Distributed Access Control. In: Proceedings of CSE 2009, vol. 3, pp. 252–258. IEEE Computer Society (2009)

25. Trivellato, D., Zannone, N., Etalle, S.: GEM: a Distributed Goal Evaluation Algorithm for Trust Management. Journal of Theory and Practice of Logic Programming (2012) (to appear)
26. Wang, Q., Jin, H., Li, N.: Usable Access Control in Collaborative Environments: Authorization Based on People-Tagging. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 268–284. Springer, Heidelberg (2009)
27. Wijesekera, D., Jajodia, S.: Policy algebras for access control - the predicate case. In: Proceedings of CCS 2001, pp. 171–180. ACM (2001)
28. Zhang, N., Ryan, M., Guelev, D.P.: Synthesising verified access control systems through model checking. Journal of Computer Security 16(1), 1–61 (2008)

Enabling Dynamic Security Policy in the Java Security Manager

Fabien Autrel, Nora Cuppens-Boulahia, and Frédéric Cuppens

Telecom-Bretagne, 35576 Cesson Sévigné, France
{fabien.autrel,nora.cuppens,frederic.cuppens}@telecom-bretagne.eu

Abstract. The Java execution environment includes several security mechanisms. They are found in the language itself, in the class loader, in the class verifier and in the sandbox in which bytecode is executed. The sandbox isolates the executed bytecode from the host on which the Java Virtual Machine (JVM) is executed. The security policy enforced by the sandbox can be configured depending on who runs a program and the origin of the program and offers fine-grained mechanisms to control resource access. However the security policy language offers no higher-level paradigms, such as the abstraction of users into roles, to enable the management of Java security policies into large infrastructures. Moreover those policies are static and cannot change depending on the state of the environment into which they are deployed. We propose in this article an approach to use the OrBAC model to configure the sandbox security policy, allowing the use of an implementation-independent policy language which offers facilities to manage large sets of JVMs, enables the expression of dynamic security policies and offers an advanced administration model.

Keywords: security policy, JVM, OrBAC, automatic configuration.

1 Introduction

The Java security model relies on several mechanisms. The Java language itself provides strong type checking, a garbage collector and access control to class members and methods. The Java Virtual Machine (JVM) also implements a class loader and a class verifier which checks various properties of the loaded bytecode. The component which isolates the JVM from the operating system in a sandbox is called the security manager. The security manager handles the external boundary of the JVM, it controls how code executed by the JVM interacts with resources outside of the JVM. This security manager is configured by a security policy specified in a file loaded when an instance of the JVM is created. This security policy is static and expressed into a language specific to the JVM which offers many low level security mechanisms. For example access control can be done on, among other resources, the file system, network connections, thread management and the Abstract Window Toolkit (AWT) framework. If we consider the fact that the security policy language offers no mechanisms to

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 180–193, 2013.

manage the expression and deployment of policies on large infrastructures, configuring this security policy for one user on a given machine is feasible but doing it for hundreds of users with various profiles on hundreds of machines becomes a tedious and error prone task.

Given the increasing size of infrastructures into which more and more complex information systems are integrated, system administrators must often configure the security of a wide range of components using ad-hoc configuration languages. This requires them to learn many languages and prevent them from having a global view on the whole system security policy. We believe that the administrators should use ideally one language to express the security policy of a whole system, which should then be enforced through a set of translators that generate configuration files for all the components.

The OrBAC [1] model attempts to address this problematic by offering several abstract concepts such as abstract entities and contextual security rules which can be used to express a dynamic security policy independently from its implementation. Moreover The OrBAC model offers a framework to analyze and solve rules conflicts, which is impossible to do using multiple security policy languages. The OrBAC model also features an administration model, the AdOrBAC [2] model, which can be used to decentralize security policy administration.

We present in this paper an approach to express and enforce dynamic security policies for the JVM security manager. Section 2 presents the existing work related to the expression and management of JVM security policies. Section 3 presents the JVM security policy language and how it is used. Section 4 presents the OrBAC model. Section 5 shows how the OrBAC model can be used to express JVM security policies. Section 6 illustrates how policies expressed in OrBAC can be enforced in a JVM. Section 7 concludes this paper and presents future works.

2 Related Work

To our knowledge there are few proposal that address the problems related to the expression of JVM security policies. The standard Java Runtime Environment (JRE) includes *Policy Tool*, a very simple application which generates JVM security policies. This application aims at making the policy specification easier by providing a user interface instead of editing directly policies in a text editor. In [3] the authors define a new policy model which includes both positive and negative authorizations. The authors use those two kinds of authorizations to define exceptions, which are not supported in the standard JRE policy language. They also define constraints as temporal constraints exclusively, enabling them to associate temporal conditions to the policy. The concept of permission delegation is also introduced in the model. Since the authors focus on the use of Java in distributed systems by using the Jini[4] framework, the notion of delegation is here restricted to the case of two JVMs communicating through the network and exchanging permissions.

However the only constraint on multi-step delegation is the delegation duration time, not the depth of a delegation chain. The authors also propose to use

the notion of groups, a group being a set of principals. This group concept is close to the notion of role in RBAC[5] or OrBAC. The policies are expressed in XML. Although this work addresses many problems in the expression and in the management of JVM security policies, the fact that it focuses on distributed systems and that the security model is an extension of the existing one makes the contribution less interesting in the context of large infrastructure administration. Actually a system administrator still has to use the new ad-hoc language to configure the JVM security policy and has no global view over the security policy of the whole system.

In [6] the authors use an Authorization Specification Language (ASL [7]) to express security policies for mobile devices. The implementation is done using a modified JRE security manager which parses XACML policies translated from the ASL representation. Unfortunately the authors only present the XACML format and not the abstract policy expressed in ASL.

In [8] the authors address the lack of flexibility of the Java 2 Micro Edition (J2ME) security model. They extend the J2ME security model and use the Security Policy Language (SPL) language [9] to express the security policy enforced on mobile devices. The J2ME is only used on mobile devices and does not use the security manager of the standard JRE so this work is not applicable in our context. The use of SPL offers policy administrators a wider view of the security deployed on a mobile device.

In [10], the authors present an approach to express firewall security policies using the OrBAC model to translate them into native firewall configuration languages. The model is independent from the targeted firewall implementation. The authors choose to represent each firewall by an organization, each firewall defining its own security policy through the specification of abstract rules in the corresponding organization. Although this paper focuses on the abstraction of firewall policy and does not address JVM policy modeling, we adopt a similar approach by defining the security policy enforced in sets of JVM hosts into organizations (cf section 5.2).

3 JVM Security Policies

The current security model implemented by the JVM security manager relies on different security mechanisms. In this paper we focus on the closed security policy which defines the sandbox boundaries. This policy specifies the permissions granted to bytecode depending on its source and the principal as which it is executed, a principal being the identity assigned to an entity, which can be the result of an authentication. This contrasts with previous JDK version before version 1.4 where access control was based only on which code is executed.

3.1 Policy Syntax

By default a single system-wide policy is defined in a file and user specific policies can optionally be defined. The system-wide policy file defines an optional

keystore entry which is used to check the public keys associated with signed bytecode. The rest of the policy file defines the permissions granted to code through the specification of "grant" entries. A grant entry specifies the sources and principals which are granted a list of permissions. The policy file syntax is defined as follows:

```
grant signedBy "signer_names", codeBase "URL",
  principal principal_class_name "principal_name",
  ... {
    permission permission_class_name "target_name", "action", signedBy "signer_names";
    ...
  };
```

Permissions have an *action* parameter which is not mandatory for all permissions. Note that the JRE security policy is static by construction, no dynamic conditions can be associated with permissions. We believe that current security requirements met by system administrators show the need for dynamic policies. System administrators should not have to learn such configuration language but should instead use higher level paradigms to express the security of an information system. Note that in this paper we do not address the security problems related to the lack of protection of a Java Runtime Environment (JRE) default installation. For example if the underlying operating system's access control mechanisms do not correctly restrict the access to a JRE setup, some of its components could be changed and/or the policy file could be easily modified [11].

3.2 Permission Types

The Java 2 security manager default implementation defines a set of permission types which define the granularity of the JVM sandbox boundary. We do not present the complete Java security policy due to space limitations and since most of the permissions types are related to very specific use cases. The use case presented in section 5 focuses on network access and file system access hence we only use the security manager $FilePermission$ and $SocketPermission$ types .

4 The OrBAC Model

The OrBAC model addresses many problems faced by system administrators in big infrastructures when specifying security policies. For example, a company infrastructure may be spread across different countries with different legislations, the employee turnover can be very high, its contractors may need to access its information system and more generally, the security policy must be modified on a regular basis. In such context, the need for multiple administrators and a dynamic security policy become central.

OrBAC aims at modeling a security policy centered on the organization which defines it or manages it. An OrBAC policy specification is done at the organizational level, also called the abstract level, and is implementation-independent. The enforced policy, called the concrete policy, is inferred from the abstract policy. This approach makes all the policies expressed in the OrBAC model

reproducible and scalable. Actually once the concrete policy is inferred, no modification or tuning has to be done on the inferred policy since it would possibly introduce inconsistencies. Everything is done at the abstract policy specification level. The inferred concrete policy expresses security rules using subjects, actions and objects. The abstract policy, specified at the organizational level, is specified using *roles*, *activities* and *views* which respectively abstract the concrete subjects, actions and objects. The OrBAC model uses a first order logic formalism with negation. However since first order logic is generally undecidable, we have restricted our model in order to be compatible with a stratified Datalog program [12]. A stratified Datalog program can be evaluated in polynomial time.

Each organization specifies its own security rules. Some *role* may have the permission, prohibition or obligation to do some *activity* on some *view* given an associated *context* is true. The *context* concept [13] has been introduced in OrBAC in order to express dynamic rules. Contexts are defined through logical rules which express the condition that must be true in order for the context to be active. In the OrBAC model such rules have the predicate *hold* in their conclusion. As suggested in [13], contexts can be combined in order to express conjunctive contexts (denoted &), disjunctive contexts (denoted $\oplus$) and context negation (denoted $\overline{ctx}$, *ctx* being a context name). Once the security policy has been specified at the organizational level, it is possible to instantiate it by assigning concrete entities to abstract entities.

5 Expressing JVM Policies in OrBAC

In this section we show how the JVM policy model can be represented using OrBAC. The main idea motivating this initiative is that ideally system administrators should be able to use the same security policy model to specify the security policy of a whole system. In the case of the OrBAC model, administrators should be able to use the same abstract entities to define the security rules which are enforced by heterogeneous security components. For example let us consider a generic *doctor* role used in a hospital policy to express the security rules common to all physicians. This role is refined by defining sub-roles such as *surgeon*, *radiologist* or *anethesist* to specify security rules that only apply to specialists. A person empowered in one of the sub-roles of the *doctor* role may access various data about his/her patients using various peripherals and applications which all must enforce the security policy. Among those applications some are executed in the Java runtime environment and others are natively executed.

In this article, we consider as a use-case a Java client application which accesses a database containing the patients medical files. Informally, the security policy associated with this application is the following: physicians can use the application, the application can open network connections to the database and can make some modifications to the local file system in order for it to correctly run.

5.1 Supported Permissions Types

As said in section 3.2, in this paper we only focus on the *FilePermission* and *SocketPermission* types. We review here in details what they represent and how they are expressed in the JVM security policy model. Although we do not model other permission types in this article, the policy translation and deployment mechanism would still be the same.

***FilePermission* Permission Type.** The *FilePermission* type represents an access to a file or directory. An instance of this permission consists of a pathname and the set of actions which can be done on the pathname. A pathname is either a file or a directory and the syntax allows the use of wild cards. The $*$ indicates all the files in a directory and $-$ indicates all the files in a directory plus recursively all files and directories contained in the directory. The possible actions on a file or directory are *read*, *write* (which implies the permission to create), *execute* and *delete*.

***SocketPermission* Permission Type.** The *SocketPermission* type represents an access to the network via sockets. An instance of this permission consists of a host specification and a set of operations which specifies how connections can be established with the host. A host name is specified as follows:

```
host = (host name | IPaddress)[:portrange]
portrange = portnumber | -portnumber | portnumber-[portnumber]
```

Four connection methods can be specified: *accept*, *connect*, *listen* and *resolve*. *resolve* is implied by the first three methods, i.e if the JVM can connect to other machines, accept connections or listen to connection then it can resolve host names. Note that this representation of network activities do not take into account network protocols and their attributes and states, limiting considerably the expression of network security policy compared to personal firewalls.

5.2 OrBAC Representation of JVM Policies

This section focuses on the definition of OrBAC abstract entities (organizations, roles, activities, views, contexts, rules) necessary to model the file system and network permissions of the Java security policy model.

Organizations. In the context of this article, a possible modeling choice would be to represent each machine running the JVM by an organization but this would possibly lead to a huge number of organizations, making the specification of JVM policies more complicated and error prone than the manual configuration of the machines. We argue here that a set of machines running the same Java applications can be abstracted into one organization in which the common security policy is defined. For example if we consider tablet PCs used by physicians when visiting their patients in a hospital, we can assume that they will all be running the same set of Java applications, or at least they can be grouped in sets of machines running the same applications. We use organization attributes to infer

the set of JVM hosts on which a JVM security policy must be deployed. More precisely, let us consider a set S_{vm1} of machines hosting the JVM on which a set of Java applications will be run. The predicate jvm_target is used to associate each element of S_{vm1} with an organization O_{vm1} modeling this set.

To avoid defining several times the same subset of the security policy in different organizations, an organization hierarchy should be defined. This way common security rules can be defined in the super organization of organizations representing different machine sets sharing some common Java application(s). We chose to define a default JVM organization as the root of the hierarchy representing the sets of machines running Java applications. Figure 1 shows an example of such hierarchy defined using the MotOrBAC[14] tool where O_{vm1} and O_{vm2} represent two subsets of machines sharing some common applications for which the policy is defined in $O_{commonApps}$. MotOrBAC is a security policy editor which implements the OrBAC model.

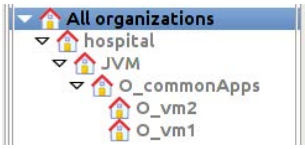


Fig. 1. A simple example of an organization hierarchy defined in a hospital

Roles. In our approach we do not define specific roles linked to the specification of JVM security policies. The roles are completely defined by the use case for which a security policy is specified. However since a Java application may send and receive network packets, some activities, defined further below, model those operations. We do not propose a new model from scratch for the modeling of network operations but rather use the approach defined in [10]. Actually we do not follow exactly the same semantic regarding the modeling of network traffic direction. In [10] a role models a machine sending network packets to a machine modeled as a view. This requires to create views corresponding to some roles to be able to specify traffic going from and to a machine. In our case the roles model users using a Java application running on a machine which sends and receive traffic. We do not create corresponding views for each role to be able to express the security policy for incoming traffic but choose to encode the network traffic direction in the activities modeling the traffic emission and reception.

Activities. As said previously, we only model in this paper the $FilePermission$ and $SocketPermission$ types. The actions defined by the $FilePermission$ type are already very generic operations that do not need much abstraction to define the corresponding activities. In fact in our experience, we noticed that the *read*, *write* and *delete* activities which abstract actions consisting in accessing various data storage entities are often present in OrBAC policies. The action of executing something is more specific to the use of software and is also easy to abstract

into the *execute* activity. We choose to model the *FilePermission* actions by making the hypothesis that in a super-organization of the *JVM* organization defined previously, generic *read*, *write*, *delete* and *execute* activities are defined. A super-activity of *read*, *write* and *delete* can be modeled as the *handle* activity, thus simplifying the specification of file system security rules for a JVM.

Regarding the modeling of the *SocketPermission* type, we follow the approach presented in [10] to specify network security policies with OrBAC, i.e activities are seen as abstraction of network services. The activity hierarchy defined in [10] consists of activities *all_protocols*, *tcp*, *udp* and *icmp*. However we have seen in section 5.1 that the *SocketPermission* type offers coarse granularity and do not take into account network protocols. We define a sub-activity of the *all_protocols* activity called *bidirectionnal* defined in the *JVM* organization and two sub-activities of the *bidirectionnal* activity, *send* and *receive*, also defined in the *JVM* organization. This way we modify locally, i.e in the *JVM* organization, the semantic of the activity modeling proposed in [10] but retain the ability to use the structure of views related to the firewall security policy. This allows us to use the aforementioned approach along with our approach to integrate the security policy specification of JVM hosts inside a more global security policy. The *SocketPermission* type does not allow to use the network protocol type to express the network security policy but port numbers can be used to identify network services. Actions considered as the *bidirectionnal*, *send* and *receive* activities have a *port* attribute which expresses a port or a port range. For example the following assertions represent two actions modeling the *ssh* and *mysql* services:

$action(ssh).\ action(mysqlV5).\ port(ssh, 22).\ port(mysqlV5, 3306).$

Port ranges are expressed the same way they are in the *SocketPermission* type syntax.

Views. To model the *FilePermission* type, the set of views to be defined and their hierarchies depend mainly on the design of the Java applications the subjects will use. We choose to represent files and directories by objects having a *path* attribute expressing the target path. Such an attribute is modeled using the following predicate:

$targetPath(obj, path)$

This models the fact that object *obj* has a target path represented by *path*. We use the same syntax as the *FilePermission* type to express the path. For example to represent a directory *application/log* located in the current user home directory by an object called *fooDir*, the following assertion would be true:

$targetPath(fooDir, \${user.home}\${/}application\${/}log)$

This object would then be used in a view representing a set of directories or files belonging to some applications having the same right on them. Here the ${/} substring is a platform-independent representation of the file separator.

Views for the *SocketPermission* type are defined in a similar way as in [10]. Such views represent sets of network machines, identified by their address or name. For example the *toDatabase* view can be defined to represent a set of

machines on which databases are installed. Objects representing network machines have an *address* attribute which represents the machine ip address or network name. View definitions can be used to manage large sets of machines. Indeed, instead of manually inserting large numbers of objects into views, view definitions can be used to automatically insert objects depending on the network address. For instance the following view definition, defined in the *hospital* organization of the example used earlier, says that an object representing a network machine H is used in the *toDatabase* view representing databases if H is part of some subnet and that it is not used in the *toBackup* view which models network backup hosts:

$use(hospital, H, toDatabase)$:-
$networkAddress(H, A)$,
$subNet(A, 10.0.0.0, 24)$,
$\neg use(hospital, H, toBackup)$.

Using the same model as in [10] allows system administrators to use the same views that have been defined when expressing the network security policy in OrBAC, thus giving administrators a more global view of their security policy.

Contexts. In section 3.1, we saw that the JVM security manager can grant rights to code depending on the location from which the bytecode is fetched (from the local file system or the network) and the identity of the subject who has signed the bytecode. This feature can be modeled by OrBAC contexts: we define two contexts types corresponding to the two conditions.

The first context type, named *codeBase*, models the code source condition expressed by the *codeBase* keyword in the JVM security policy syntax. The following derivation rule shows an example of such context:

$hold(JVM, S, \_, \_, codeBaseFoo)$:-
$byteCode(S, B)$,
$codeBase(B, database.intranet.mycompany.com)$.

This context is true in organization JVM if the bytecode executed by the subject S has been downloaded from a server *database.intranet.mycompany.com* in the intranet of some company.

The second context type, named *signedCode*, models the code source condition expressed by the *signedBy* keyword in the JVM security policy syntax. The following derivation rule shows an example of such context:

$hold(JVM, S, \_, \_, signedCodeFoo)$:-
$byteCode(S, B)$,
$signedBy(B, peter)$.

This context is true in organization JVM if the bytecode executed by the subject S has been signed by Peter. The JVM security policy syntax support the specification of code signed by multiple subjects, which can be easily taken advantage of in our modeling. For example the following context models a condition where at least one of the developers of some bytecode must have signed it in order to be true:

$hold(JVM, S, \_, \_, signedCodeFoo)$:-
$byteCode(S, B)$,
$developpedBy(B, D)$,
$signedBy(B, D)$.

Here the $developpedBy(b, d)$ predicate is true if bytecode b has been developed by subject d.

Those contexts can be defined and used in the specifications of security rules inside the JVM organization.

Security Rules. In our approach we use the standard implementation of the JVM security manager, which implements a closed policy. Hence in this paper the translation process only translates permissions specified in the JVM organization and its sub-organizations. The JVM security policy is specified inside the organization hierarchy defined in the JVM organization using the roles defined in the super organizations of JVM. The activities, views and contexts defined previously are used to define all the JVM abstract permissions.

However the system administrators are not limited to the use of the previously defined context types when specifying the abstract permissions. As said in the introduction of this paper, JVM security policies are static. In our approach any other OrBAC context type can be used to make the policy dynamic. Actually contexts can be composed using conjunction and disjunction operators to associate complex contextual conditions with permissions. When a context state changes for some concrete entity triple $\{subject, action, object\}$ in the JVM organization or one of its sub-organizations, the new inferred concrete policy is pushed on the hosts specified in the corresponding organization attributes. Such context can be, but is not limited to, a temporal context, a spatial context expressing a condition on the position of a subject in space, a condition on some concrete entity attribute or a condition on the system state. Since the JVM security manager standard implementation does not refresh the security policy if the policy file is modified after a JVM instance is started, we have modified the standard implementation to trigger this refreshment.

6 Enforcing JVM OrBAC Policies

In this section we present the OrBAC JVM policy to JVM policy translation algorithm and illustrate it with an example and an implementation. The translator which also updates the security policy files on target hosts running a JVM is implemented as a MotOrBAC plug-in. MotOrBAC is used to specify the abstract security policy and associate concrete entities with abstract entities. It is also used to specify the list of hosts on which the security policy must be deployed.

6.1 Translation Algorithm

The algorithm does not translate the abstract security policy but rather the concrete security policy which is inferred by the OrBAC Application Programming

Interface (API) inference engine. The OrBAC API is used by MotOrBAC to process OrBAC policies. The concrete permissions inferred by the OrBAC API have many attributes like the contexts to which they are associated and the organization in which they have been inferred. Each inferred concrete permission is parsed to generate a grant entry. The JVM security policy syntax does not support the specification of different policies for different users in one file. Hence the translation process generates one policy file per subject.

Let us consider a subject for which a set P of concrete permissions related to a JVM security policy has been generated. For each permission p in P, the translation process generates a grant entry for each contextual condition on the origin of the code. Then for each of those entries, the list of signers are added if the contextual condition contains such condition. The type of permission to add to the grant entry and its attributes are extracted from the parameters of p.

When the list of permissions for a subject changes because some contexts have been activated or deactivated, the corresponding security policy file is generated and pushed on the hosts the user may use.

6.2 Example

We consider an example based on the one presented in section 5.2 of an OrBAC policy specified in a hospital. We assume that physicians use tablet PCs when they visit their patients to access their files. The client application is a Java applet, which must be signed by the main developer *bob*, running inside a web browser which connects to a database where patient files are stored. The applet can connect to the database but not the opposite. The applet used by physicians uses a directory structure created in the user home directory. This directory is called *appletDir* and contains three other directory storing specific files: the *resource* directory, which can only be read, the *log* directory, which can only be written and the *temp* directory which can be read and written. We assume that the OrBAC policy is already structured according to the roles defined in the hospital and that a network security policy has been defined according to the approach in [10]. Hence we assume a *doctor* role has been defined in organization *hospital*. We define a sub-organization of the JVM organization, called *appletOrg*, in which the security policy applied to peripherals running the applet is deployed. The list of hosts on which the policy is deployed is specified in the *appletOrg* organization.

The technical details for the considered use case are the following:

- a DNS server is used in the private network
- the database server and the client Java applet are run on Linux machines
- the database is a Mysql 5 database listening on port 3306 and hosted on machine *database.intranet.hospital.com*
- the web page from which the applet is retrieved is: *http* : *//applet.intranet.hospital.com*

A *mysql* action models the action of using the Mysql database. Its *port* attribute is set to 3306. It is considered as the *send* activity in the JVM organization

because the applet connects to the database but does not accept connections. Two actions *readFilesystem* and *writeFilesystem* are considered respectively as the *read* and *write* activities in the *JVM* organization.

We assume that the database server has already been modeled by a view *database* in the network related part of the OrBAC policy. The object *db*1 is used in this view in the *hospital* organization. Its *address* attribute is set to the host name specified above. Three views are defined to model the applet directories: *resource*, *log* and *temp*. Three objects *resource_applet*1, *log_applet*1 and *temp_applet*1 are used respectively in the *resource*, *log* and *temp* views in the *JVM* organization. Their *targetPath* attribute is set respectively to ${*user.home*}${/}*appletDir* ${/}*resource*, ${*user.home*}${/}*appletDir*${/}*log* and ${*user.home*}${/} *appletDir*${/}*temp*.

We define a *codeBase* context to model the condition on the applet source bytecode:

$hold(appletOrg, S, \_, \_, cbCtx)$:-
$byteCode(S, B)$,
$codeBase(B, applet.intranet.hospital.com)$.

The following *signedBy* context models the condition on the applet signed bytecode:

$hold(appletOrg, S, \_, \_, scCtx)$:-
$byteCode(S, B)$,
$signedBy(B, bob)$.

We also define a *visitTime* temporal context in the *hospital* organization which is only active when doctors are visiting their patients. Using the previously defined abstract entities we can write the abstract permissions corresponding to the example:

$permission(appletOrg, doctor, send, database, scCtx\&cbCtx\&visitTime)$
$permission(appletOrg, doctor, read, resource, scCtx\&cbCtx)$
$permission(appletOrg, doctor, write, log, scCtx\&cbCtx)$
$permission(appletOrg, doctor, handle, temp, scCtx\&cbCtx)$

Assuming that a subject *daniel* is empowered in the doctor role in the *hospital* organization and that the *visitTime* is active for *daniel* in the *hospital* organization, the following concrete permissions are inferred:
$permission(daniel, mysql, db1)$
$permission(daniel, readFilesystem, resource\_applet1)$
$permission(daniel, writeFilesystem, log\_applet1)$
$permission(daniel, readFilesystem, temp\_applet1)$
$permission(daniel, writeFilesystem, temp\_applet1)$

6.3 Implementation

We have developed a MotOrBAC plug-in implementing the translation process and the configuration of the JVM hosts. Four virtual machines have been created to represent the database server, a tablet PC running the applet, a web server from which the applet is downloaded and the administrator host running MotOrBAC and the plug-in. Generated policy configuration files are uploaded by the

plug-in into the users home directory through ssh connections using public key authentication. The list of hosts to which the files are transfered is inferred from the *appletOrg* organization attributes as specified in section 5.2. Configuration files are transfered to the hosts whenever a change in contexts state have been triggered. We have modified the standard JVM security manager implementation to reload the security policy while a JVM instance is executed when a change is detected in the policy file. This way JVM security policies are dynamically updated as the concrete policy evolves in time.

From the concrete permissions inferred in the previous section and the concrete entities attributes, a JVM security policy configuration files is generated:

```
grant signedBy "bob", codeBase "http://applet.intranet.hospital.com" {
  permission Java.io.FilePermission "\${user.home}\${/}appletDir\${/}resource", "read";
  permission Java.io.FilePermission "\${user.home}\${/}appletDir\${/}log", "write";
  permission Java.io.FilePermission "\${user.home}\${/}appletDir\${/}temp", "read,write";
  permission Java.net.SocketPermission "database.intranet.hospital.com:3306", "connect";
};
```

Note that generated grant entries having the same *signedBy* and *codeBase* conditions are grouped to generate smaller files. We use the *rsync* program to generate less network traffic when uploading the configuration files. Note that although we have used only Linux machines in our proof of concept, the generated policy files could directly be used on other operating systems as we used generic variables to identify the current user home directory and the file system separator.

7 Conclusion

In this article we presented an approach to abstract the JVM security policy model into the OrBAC model and a proof of concept using the MotOrBAC tool. This allows system administrators to use a powerful dynamic security model to express the security requirements applied to JVM instances instead of applying the ad-hoc policy language defined in the standard JRE. We think that the main advantage of this approach is that system administrators can use the same model and the same abstract entities to define security policies applied to heterogeneous security components, thus giving them a global view of their information system without having to specify separate policies in different languages for each component. Moreover the dynamic nature of OrBAC policies and the dynamic deployment of configuration files implemented in our approach provides means to change the security properties of running Java applications, which is not possible for a standard JRE.

Another main advantage of this approach is that system administrators can use the AdOrBAC [2] model to administrate the specification of JVM policies in OrBAC. MotOrBAC implements the AdOrBAC model, including the delegation model, which means that administration tasks can be managed. For example a system administrator can delegate to another subject the right to define only permission related to the network policy of JVMs.

Using the standard security manager implementation limits the granularity of the policies we can express, especially regarding the network policies. We plan

to modify the security manager implementation to refine its boundaries and directly integrate the OrBAC API inside it. This way OrBAC policies specified with MotOrBAC could be directly interpreted without the need for a translator.

Acknowledgements. This research has been supported by the European Commission and the ANR, respectively in the framework of the ITEA2 Role-ID project (Grant agreement no.08007) and the SELKIS project (ARN ARPEGE project).

References

1. Kalam, A.A.E., Baida, R.E., Balbiani, P., Benferhat, S., Cuppens, F., Miège, Y.D.A., Saurel, C., Trouessin, G.: Organization based access control. In: IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Policy 2003 (2003)
2. Cuppens-Boulahia, N., Cuppens, F., Coma, C.: Multi-granular licences to decentralize security administration. In: First International Workshop on Reliability, Availability, and Security (WRAS), Paris, France (2007)
3. Samson, F.: Alternative Java Security Policy Model. Phd. thesis, Université Laval (2004)
4. River, A.: Jini: a network architecture for the construction of distributed systems (2010), `http://river.apache.org`
5. Ferrailo, D.F., Sandhu, R., Gavrila, S., Kuhn, D., Chandramouli, R.: Proposed NIST standard for rbac. ACM Transactions on Information and System Security (2001)
6. Zhang, X., Parisi-Presicce, F., Sandhu, R.: Towards remote policy enforcement for runtime protection of mobile code using trusted computing (2006)
7. Jajodia, S., Samarati, P., Subrahmanian, V.S., Bertino, E.: A unified framework for enforcing multiple access control policies. In: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, SIGMOD 1997, pp. 474–485. ACM, New York (1997)
8. Dragovic, I.I.B., Crispo, B.: Extending the java virtual machine to enforce fine-grained security policies in mobile devices. In: Proceedings of the Annual Computer Security Applications Conference, ACSAC (2007)
9. Ribeiro, C., Zúquete, A., Ferreira, P., Guedes, P.: Spl: An access control language for security policies with complex constraints. In: Proceedings of the Network and Distributed System Security Symposium, pp. 89–107 (1999)
10. Cuppens, F., Cuppens-Boulahia, N., Sans, T., Miège, A.: A formal approach to specify and deploy a network security policy. In: Second Workshop on Formal Aspects in Security and Trust, FAST (2004)
11. Wheeler, D., Conyers, A., Luo, J., Xiong, A.: Java security extensions for a java server in a hostile environment. In: Proceedings of the 17th Annual Computer Security Applications Conference, ACSAC 2001, p. 64. IEEE Computer Society, Washington, DC (2001)
12. Ullman, J.D.: Principles of database and knowledge-base systems. Computer Science Press (1989)
13. Cuppens, F., Cuppens-Boulahia, N.: Modeling contextual security policies. International Journal of Information Security (IJIS) 7(4) (August 2008)
14. Autrel, F., Cuppens, F., Cuppens-Boulahia, N., Coma, C.: Motorbac 2: a security policy tool. In: Third Joint Conference on Security in Networks Architectures and Security of Information Systems, SARSSI (2008)

A Novel Obfuscation: Class Hierarchy Flattening

Christophe Foket*, Bjorn De Sutter, Bart Coppens, and Koen De Bosschere

Computer Systems Lab
Electronics and Information Systems Department
Ghent University
Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium
{cfoket,bcoppens,kdb,brdsutte}@elis.ugent.be

Abstract. This paper presents class hierarchy flattening, a novel obfuscation technique for programs written in object-oriented, managed programming languages. Class hierarchy flattening strives for maximally removing the inheritance relations from object-oriented programs, thus hiding the overall design of the program from reverse engineers and other attackers. We evaluate the potential of class hierarchy flattening by means of a fully automated prototype tool for Java bytecode. For real-life programs from the DaCapo benchmark suite, we demonstrate that the transformation effectively hinders both human and tool analyses, and that it does so at limited overheads.

Keywords: Java bytecode, obfuscation, class hierarchy, program design.

1 Introduction

Reverse engineering and modification of managed code are well-understood and common practices, with many legitimate goals [16]. Malicious developers can abuse them, however, to attack Java and .NET applications with the goals of software piracy, software IP theft, and data theft. Their attacks are facilitated by the fact that managed code is executed at a high abstraction level. To combine run-time efficiency with programmer productivity, a large amount of symbolic information needs to be presented to the virtual machines that execute the code. This is needed, e.g., to enable effective and efficient just-in-time (JIT) compilation, to support efficient garbage collection, and to support reflection and bytecode verification. This symbolic information is also what makes managed code easier to understand, reverse engineer, decompile, modify, reuse and steal.

With respect to reverse engineering (and all practices for which reverse engineering is a prerequisite), many different obfuscation techniques have been proposed. Some try to prevent automatic decompilation [5,17], some try to hide data (flow) properties [8,34] or control flow properties [8,9,17,19,20,22,24,33]

* The authors want to thank the Agency for Innovation by Science and Technology in Flanders (IWT) for their support and Ghent University, the Hercules Foundation and the Flemish Government - department EWI who funded the STEVIN Supercomputer Infrastructure at Ghent University on which we carried out part of this work.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 194–210, 2013.

from both tools and humans. Others try to remove information that is useful informally, such as field and method identifiers [2,5]. Finally, a few have proposed obfuscating the overall application design by altering the class and interface hierarchy to make it harder to understand for software engineers [28]. The latter techniques aim for the opposite of classic code refactoring [27,31].

This paper takes application design obfuscation one step further. Instead of merely modifying an application's type hierarchy, we propose a technique called class hierarchy flattening (CHF) to get rid of it altogether. Given a number of constraints because of, e.g., compatibility with external libraries, CHF strives for a class hierarchy that is as flat as possible, i.e., in which application classes are siblings rather than subtypes and supertypes. We discuss the necessary analyses and transformations to automate CHF and present a proof-of-concept tool. We evaluate the level of software protection provided by CHF and its overhead.

The remainder of this paper is structured as follows. First, Section 2 discusses the conceptual goals of CHF by means of an example program. The transformation itself is discussed in some detail in Section 3, and evaluated in Section 4. We compare CHF to related work in Section 5. Finally, Section 6 draws conclusions and discusses some future extensions.

2 Rationale: An Example Program

To set the context for a detailed discussion of our obfuscating class hierarchy transformation, we first present an example consisting of a media player. It consists of three main parts: (1) the player initializer (2) support for media files and (3) support for media streams contained in the media files. Different subtrees of the class hierarchy implement those parts, as shown in Figure 1. The code in Figure 2(a) illustrates their interaction.

The `main` method of class `Player` creates an array of `MediaFile` objects to be played (line 10). It then queries each of the media files in this list for its media streams (line 12), which are initialized when the media file is accessed with the `readFile` method. Figure 2(a) shows how this is done for the `MP3File` class, which represents MP3 files containing MPEG audio streams.

During playback, the player checks the run-time type of the `MediaStream` object associated with the stream (lines 13 and 15) to decide where it needs to be output. Depending on the actual run-time type of the `MediaStream` objects, they are either cast to `AudioStream` or `VideoStream`, such that the correct `play` method is invoked (lines 14 and 16). The `play` methods essentially output the raw bytes of the media streams' analog signals for a specific output device. Those bytes are obtained, decrypted (lines 35–36) and decoded (line 37) with the `getRawBytes` method declared in `MediaStream`. Note that because the decoding process is different for each type of media stream, the `decode` method is declared as abstract, such that it can be implemented by subclasses of `MediaStream`. The decryption process, on the other hand, is the same for each type of media stream and is therefore handled by the `MediaStream` class.

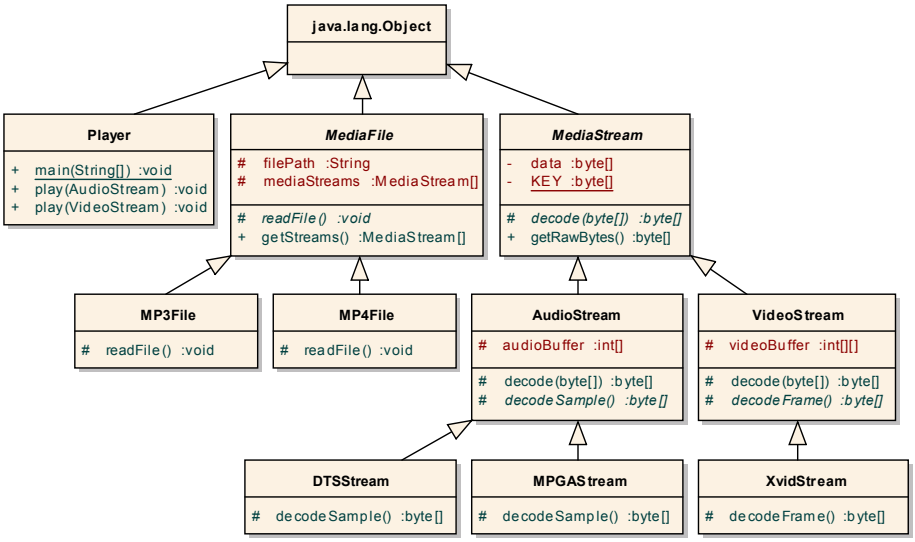


Fig. 1. Class hierarchy of a simple DRM media player

From a software-engineering point of view, the media player application is well structured. The inheritance relations are meaningful and code shared between different classes is located in a common superclass. While we could have further improved the structure of the program by factoring out the casts and run-time type checks, we chose not to do so for educational purposes.

From a security perspective, however, some problems arise. First, the class hierarchy provides reverse engineers with information about the relationships between classes and the abstraction levels of the functionalities provided by classes (with classes higher in the hierarchy typically providing more abstract functionality). Secondly, code sharing through inheritance enables attacks in which compromising one class can cause all of its subclasses to be compromised. All media streams are decrypted using the `getRawBytes` method declared in `MediaStream`. Therefore, when an attacker reverse-engineers this method, he will be able to decrypt all supported media stream types. Finally, we observe that the program contains much type information, which simplifies both manual analysis and automated analysis that rely on, a.o., point-to set computations.

These issues can be solved by rewriting the well-structured hierarchy into the unstructured class collection of Figure 3. To determine how classes are related, an attacker can then no longer rely on the class hierarchy. He will instead have to analyze and compare all classes in the application. Furthermore, as all classes are provided with a (diversified) copy of all fields and methods declared in their former superclasses, they have become functionally more independent. Code is no longer shared between related classes, so attackers can no longer attack many classes at once by patching their common superclass. In short, more actual code analysis and tampering will be needed to mount a successful attack.

```
1  public class Player{
2    public void play(AudioStream as) {
3      /* send as.getRawBytes() to audio device */
4    }
5    public void play(VideoStream vs) {
6      /* send vs.getRawBytes() to video device */
7    }
8    public static void main(String[] args) {
9      Player player = new Player();
10     MediaFile[] mediaFiles = ...;
11     for(MediaFile mf : mediaFiles) {
12       for(MediaStream ms : mf.getStreams())
13         if(ms instanceof AudioStream)
14           player.play((AudioStream)ms);
15         else if(ms instanceof VideoStream)
16           player.play((VideoStream)ms);
17     }
18   }
19 }
20
21 public class MP3File extends MediaFile {
22   protected void readFile() {
23     InputStream inputStream = ...;
24     byte[] audioData = new byte[...];
25     inputStream.read(audioData);
26     AudioStream as = new MPGAStream(audioData);
27     mediaStreams = new MediaStream[]{as};
28   }
29 }
30
31 public abstract class MediaStream {
32   public static final byte[] KEY = ...;
33   public byte[] getRawBytes() {
34     byte[] decrypted = new byte[data.length];
35     for(int i = 0; i < data.length; i++)
36       decrypted[i] = data[i] ^ KEY[i];
37     return decode(decrypted);
38   }
39   protected abstract byte[] decode(byte[] data);
40 }
```

(a) original code

```
1  public class Player implements Common {
2    public void play(Common as) {
3      /* send as.getRawBytes() to audio device */
4    }
5    public void play1(Common vs) {
6      /* send vs.getRawBytes() to video device */
7    }
8    public static void main(String[] args) {
9      Common player = new Player();
10     Common [] mediaFiles = ...;
11     for(Common mf : mediaFiles) {
12       for(Common ms : mf.getStreams())
13         if(myChecker.isInstance(0, ms.getClass()))
14           player.play(ms);
15         else if(myChecker.isInstance(1, ms.getClass()))
16           player.play1(ms);
17     }
18   }
19 }
20
21 public class MP3File implements Common {
22   public void readFile() {
23     InputStream inputStream = ...;
24     byte[] audioData = new byte[...];
25     inputStream.read(audioData);
26     Common as = new MPGAStream(audioData);
27     mediaStreams = new Common []{as};
28   }
29 }
30
31 public class MediaStream implements Common {
32   public static final byte[] KEY = ...;
33   public byte[] getRawBytes() {
34     byte[] decrypted = new byte[data.length];
35     for(int i = 0; i < data.length; i++)
36       decrypted[i] = data[i] ^ KEY[i];
37     return decode(decrypted);
38   }
39   public abstract byte[] decode(byte[] data);
40 }
```

(b) flattened code

Fig. 2. Partial implementation of the `Player`, `MediaStream` and `MP3File` classes

Code analysis has also become harder, as the code in the transformed application (shown in Figure 2(b)) contains less type information than the original application. This follows from all declaration types being replaced by a new `Common` type. Since this interface type serves as a common supertype for all classes in the application and declares all their instance methods, all classes implement a significantly larger number of methods. The subset of those methods that are never called at run time can be filled in with arbitrary code, to make the static analysis of the application even more complex.

3 Class Hierarchy Flattening

This section presents the analyses and transformations needed to automatically transform the unprotected program into the one that is much harder to attack.

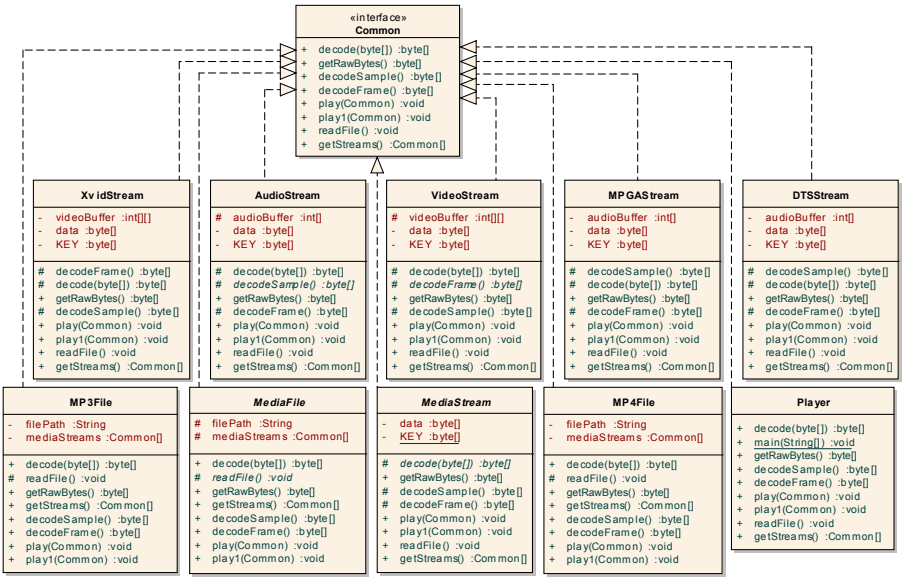


Fig. 3. Flattened class hierarchy of the media player

3.1 Basic Algorithm

The basic class hierarchy flattening (CHF) algorithm consists of five steps.

Step 1: Subtree selection. We assume that each application consists of a set of *application classes* $\mathbb{A}$ that use or extend classes from a self-contained set of *library classes* $\mathbb{L}$ that includes `java.lang.Object`. Classes in $\mathbb{L}$ are never considered for transformation. $\mathbb{L}$ will usually correspond to the standard library, while $\mathbb{A}$ will contain all classes that make up the actual application. In this paper, $\text{sub}(x)$ denotes all subclasses of class x, and $\text{super}(x)$ its superclasses.

There is a subset $\mathbb{X} \subset \mathbb{A}$ of classes on which our CHF transformation is not applicable because changing those classes' position in the hierarchy can alter the program behavior. This includes classes on which reflective operations are performed such as `getInterfaces()` (which makes the program potentially dependent on the number of interfaces implemented by a class), `getSuperclass()`, `isAssignableFrom()`, `getMethod()`, etc.

As library classes cannot be rewritten, we cannot change their position in the hierarchy, nor can we adapt their methods' signatures, which typically involve library types themselves. To maintain type correctness, this implies that in general any application class $a \in \text{sub}(l)$ with $l \in \mathbb{L}$, needs to stay a subclass thereof. This further implies that we cannot make all application classes direct subclasses of `java.lang.Object`. This limitation is similar to the limitations imposed to several code refactoring transformations. Those limitations have been formalized in literature [31], so we will not repeat them here. As the classes in $\mathbb{X}$ cannot be moved in the hierarchy either, similar limitations apply to them.

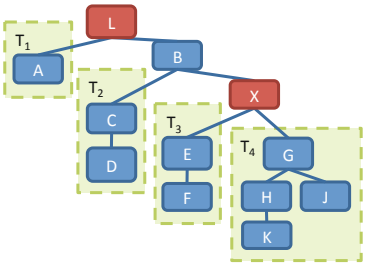


Fig. 5. Selected subtrees

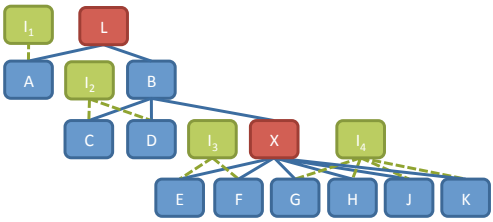


Fig. 6. Flattened class hierarchy subtrees

$$\mathbb{T} = \bigcup_{i=1..m} T_i$$

$$\forall i, j. T_i \cap T_j = \emptyset$$

$$T_i \subset \mathbb{A} \setminus \mathbb{X}$$

$$\forall c \in T_i. \text{sub}(c) \subset T_i$$

Fig. 4. Selection rules

For our purpose, we partition $\mathbb{A}$ into the set $\mathbb{T}$ of transformable classes and the set $\mathbb{N}$ of non-transformable classes. $\mathbb{T}$ is further partitioned into disjoint subtrees T_i according to the rules of Figure 4. They mainly express that each subtree T_i consists of a unique set of transformable classes for which the property holds that if T_i includes a class c, it also includes all of its subclasses. An example selection of subtrees is depicted in Figure 5. In the media player, the three subtrees correspond to the three subtrees of `java.lang.Object`.

Each subtree will then be transformed into one flat set of classes that all implement a common interface and that are all direct subclasses of the direct superclass of the tree's root. For the class hierarchy of Figure 5, the result with four new interfaces can be seen in Figure 6.

Step 2: Interface insertion. To reach that final result, we first add the common supertype interfaces in two steps. For each subtree we encapsulate all instance fields declared in all classes of the subtree with getter and setter methods and rewrite public accesses to those fields into invocations of these getters and setters. This is done to provide access to instance fields declared in the subtree classes, even though interfaces cannot declare instance fields. Next, we create a new supertype interface for each subtree. This interface declares all instance methods of all classes in the subtree and is implemented by all classes of the subtree. Whenever an original class does not implement all the required methods of the interface, temporary dummy methods are added, some of which implement supercalls not to change the behavior of overriding methods.

Step 3: Subtree type abstraction. Next, we rewrite the program such that it becomes independent of the subclass relations that will be removed from the class hierarchy. We replace all references to types in $\mathbb{T}$ by their corresponding interface supertype. In practice, this comes down to replacing the types of local variables, fields, array creations, and the types used in method signatures. The only time we still refer to the actual classes in the subtree is for object creation. An example of such a conversion of declarations can be seen in Figure 2, where various declarations have been replaced by the supertype `Common`.

As for cast operations, most of them are not needed anymore for static type correctness because interfaces instead of concrete types are used in declarations wherever possible. Moreover, we want to omit the remaining ones from the code not to reveal type information. So we replace them by code that tests a type and throws a `ClassCastException` whenever a run-time cast would have failed in the original program. To minimize the number of types that needs to be tested (and hence revealed in the code), we perform a points-to analysis on the original program [15,23]. As such, our treatment of casts is similar to that in other code refactoring techniques that change type hierarchies [31].

Step 4: Subtree flattening. Finally, we remove the inheritance relations between the classes in the subtrees, making subclasses independent of their superclasses. We traverse each subtree T_i in a breadth-first fashion. For each class `c` $\in T_i$, we execute the following steps for each direct subclass `d` of `c`:

1. copy the instance fields and concrete instance methods from `c` to `d`, renaming them if necessary to avoid collisions with original fields and methods of `d`;
2. rewrite the code in `d` such that it makes use of its own private copies of the methods and fields defined in `c`;
3. make `d` implement the same interfaces as `c`, to preserve assign compatibility between variables and fields of the interface types and objects of type `d`;
4. make `d` a sibling of `c` by setting its superclass to the superclass of `c`.

During this flattening, we replace many of the temporary dummy methods that were added when the interfaces got inserted. Not all of those methods are replaced, however. Consider, e.g., the `MediaStream` subtree. The interface for this subtree declares both the methods `decodeFrame()` and `decodeSample()`, and hence all classes originating from this subtree should implement those methods. That is why we inserted dummy implementations where necessary. In this case, some of the dummy implementations of `decodeFrame()` and `decodeSample()` in `DTSStream`, `MPGAStream`, and `XvidStream` are overwritten, but those in, e.g., `MediaStream`, `AudioStream`, and `VideoStream` are not. This poses no problem: As the non-overwritten methods were not present in the original program, and as we are not changing the behavior of the program, they will never be executed. We can therefore provide a dummy implementation for them, using nonsensical code [2] or carefully chosen code, as we propose in Section 3.2.

Step 5: instanceof. The behavior of run-time type checks, introduced either explicitly by the programmer or automatically while handling casts during subtree type abstraction, depends on the specific organization of classes in the hierarchy. Before flattening the subtrees of Figure 1, `ms instanceof AudioStream` evaluated to `true` for `ms` pointing to objects of either type `DTSStream` or `MPGAStream`. In the flattened subtree, however, it evaluates to `false` for objects of those types.

To maintain the original behavior, we replace all occurrences of `instanceof` by a lookup in a table that encodes part of the original subtype relations, namely the part that is necessary to maintain the behavior with respect to `instanceof`. Each row in the lookup table initially corresponds to one of the instanceof expression $\texttt{o}_i$ `instanceof` $\texttt{A}_j$ in the program, while the columns correspond to

Table 1. `instancoef` lookup table

	XvidStream	AudioStream	VideoStream	MPGAStream	DTSStream	MP3File	MediaFile	MediaStream	MP4File	Player
`ms instanceof AudioStream`	false	DC	DC	true	true	DC	DC	DC	DC	DC
`ms instanceof VideoStream`	true	DC	DC	false	false	DC	DC	DC	DC	DC
`mf instanceof MediaFile`	DC	DC	DC	DC	DC	true	DC	DC	true	false

(a superset of) the classes in the points-to set of all o_i. For the example program introduced in Section 2, the initial lookup table is given in Table 1.

As most of the classes will not occur in all points-to sets of all occurrences of `instanceof`, a considerable number of elements in the table will be "don't care" (DC) values. As is done for the optimization of multi-output boolean functions for optimizing integrated circuits [21], we can freely choose how to instantiate those DCs, i.e., replace them by `true` or `false`. For example, as `MPGAStream` and `DTSStream` have identical behavior, they likely originate from the same subtree. We can hide this by instantiating their DC values in a way that makes the classes' behavior look different, thus hiding an existing relation between two classes in the program. Alternatively, we can make `XvidStream` and `Player`, which are not related at all, look related by instantiating their DC values such that their behavior becomes identical. Likewise, we can replace the last two, different occurrences of `instanceof` by two identical ones by instantiating their DCs appropriately. This way, completely unrelated cast operations look as if they cast related types.

In short, by instantiating the DC values in the table, we can reduce its size and make unrelated classes and casts look related and vice versa. Furthermore, we can use hashing and white-box crypto techniques [6] to prevent static analysis of the table and involved code. Exploiting these opportunities is future work.

Once the final lookup table is constructed, each expression $\mathtt{o}_i$ `instanceof` $\mathtt{A}_j$ is replaced by a call `myChecker.isInstance(`$\mathtt{r}_{i,j}$`,`$\mathtt{o}_i$`.getClass())` where $\mathtt{r}_{i,j}$ is the row index of the lookup table entry that corresponds to the given instanceof expression.

3.2 Extensions

Several extensions to CHF can be considered.

Dummy methods - introducing differences/similarities. During the subtree flattening, methods and fields are copied from parent classes into their children. This creates an opportunity for attackers to infer the original class hierarchy by means of diffing tools like Stigmata (http://stigmata.sourceforge.jp/). To distract such tools, we can introduce artificial differences or similarities by

choosing appropriate dummy method bodies. By copying function bodies from unrelated classes, we can make unrelated classes look related and vice versa.

Interface Merging. CHF as described above binds each subtree to one interface. This gives attackers the possibility to infer information about the original class hierarchy from the use of interfaces. To limit the amount of information that can be inferred, we can merge multiple (unrelated) interfaces into a single one. Such merging can result in more dummy methods in the classes, however, and hence in considerable overhead. This can be observed in the flattened media player hierarchy in Figure 3, in which the three interfaces are already merged.

It is important to note that in general, the merging of interfaces needs to be limited to subtrees originating from within the same jar files. The merged interface can then be packaged in that same jar, such that custom class loaders, of which it is not known which jar files they can access in the original program, can find them precisely when and where they need them.

Object Allocation Obfuscation. Even after interface merging, some statements expose detailed type information. After the allocation on line 26 in Figure 2, `as` points to an object of type `MPGAStream`. From this information, points-to analysis deduce points-to sets of many local variables. In turn, other analyses like call graph construction and program slicing will also regain some precision to the advantage of attackers. To prevent the propagation of precise type information at allocation sites, we can replace individual allocations by multiple ones by means of opaque predicates [24]. For example, line 26 can be replaced by

```
Common as;
if(condition1) as = new XvidStream(...);
else if(condition2) as = new DTSStream(...);
else if(condition3) as = new MPGAStream(audioData);
else as = new AVC1Stream(...);
```

in which the first two conditions opaquely evaluate to false, and the third one opaquely evaluates to true. Switch statements can also be used of course. Alternatively, we can introduce factories [12] that return all types that implement an interface. Factories are more stealthy [7] as they look more like regular, well-engineered code. Moreover, whereas context-insensitive coverage analysis suffices to detect that potential opaque predicates or switches only evaluate to one value, context-sensitive ones will be needed to obtain equally useful information from factories implemented in separate methods.

The effect of such object allocation obfuscations, when not undone by an attacker, will be that no points-to analysis, however complicated, will compute more precise results than the analysis based on class hierarchy analysis [10].

Combining Flattening with Other Obfuscations. CHF can be combined with several existing design obfuscations. CHF enables, e.g., more efficient class coalescing. Coalescing `MP3File` and `VideoStream` in Figure 1 with the algorithm proposed by Sosonkin et al. [28] would require `MediaFile` and `MediaStream` to be coalesced as well. This would increase the number of fields in all classes that inherit from the coalesced class by a factor two. After CHF, `MP3File` and

`VideoStream` can be coalesced without affecting the number of fields, and consequently the size of objects, of other classes.

CHF can also prepare a program for false factoring [8]. In Figure 3 all classes inherit directly from `java.lang.Object` and dependencies on the original inheritance relations have already been removed, so the classes can easily be reorganized in a fake hierarchy by inserting random superclasses.

4 Evaluation

We implemented CHF in Soot 2.5.0 [18,32], an analysis and transformation framework for Java bytecode. As our tool rewrites the application bytecode that the developer has packaged in a collection of jar files, it does not require any changes to the application's source code.

Our implementation consists of two parts; a CHFTransformer and a refactoring toolkit. The CHFTransformer implements CHF as a Soot SceneTransformer, such that it can be applied together with Soot's other whole program transformations. Our refactoring toolkit provides a series of refactoring transformations, including *encapsulate field*, *rename field/method*, and variations of *push down field/method* and *extract interface* that were required to implement CHF [11].

To detect the set of non-transformable classes and to ensure that all Java features like reflection and dynamic class loading are handled correctly, we rely on TamiFlex, a tool developed specifically for facilitating the static analysis of Java programs that use such features [4]. As a profile-based tool, TamiFlex relies on the developer to provide program inputs that generate enough coverage. Alternatively, the developer can manually complement the coverage of TamiFlex with his knowledge of how the program depends on reflection and class loaders.

4.1 Benchmarks

We use the DaCapo 9.12 benchmark suite [3] to evaluate the protection-wise effectiveness and the performance-wise efficiency of CHF. This suite consists of 14 medium to large sized realistic applications. Because of space concerns, we report results for the four applications listed in Table 2. We opted for the "9.12 bach" release of the DaCapo suite because TamiFlex is particularly well tested on this version (http://dacapobench.org/soot.html). As can be seen in the table, for three out of four benchmarks the large majority of all classes is transformable. For eclipse, the number of transformable classes is much lower, because of restrictions imposed by dynamically generated classes.

For all benchmarks, we generated and evaluated $1 + 1 + 5 \times 10$ versions. The first, base version is the original bytecode that comes with the DaCapo suite, but with identifier names obfuscated [7]. This type of obfuscation is orthogonal to CHF; any Java obfuscator would apply it. We applied it for our evaluation baseline to obtain results for realistically obfuscated programs and to be able to present realistic overheads in term of code size and memory footprint, both of which heavily depend on the length of identifiers.

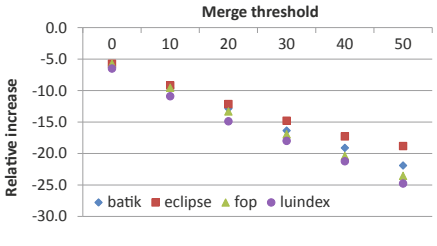
Table 2. Overview of DaCapo 9.12-bach benchmarks before and after Identifier Obfuscation (IO)

Benchmark	Description	# appl. types	# transf. classes	# jar files	code size (MB)	
					before IO	after IO
batik	Scalable Vector Graphics processor	4573	3505 (77%)	6	12.5	9.3
eclipse	non-GUI version of Eclipse IDE	5947	2258 (38%)	48	25.7	22.7
fop	XSL-FI to PDF conversion	4479	3349 (75%)	7	11.0	8.8
luindex	document indexing based on Lucene	633	526 (83%)	3	1.9	1.2

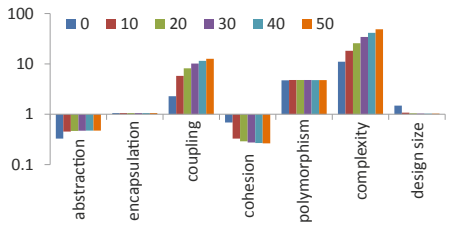
The second version was generated from the first one by our prototype implementation of the basic CHF algorithm as discussed in Section 3.1. Next, we extended the basic algorithm with interface merging (Section 3.2) and we generated program versions at different levels of interface merging. Given a merge threshold value $t \in \{10, 20, 30, 40, 50\}$, the extended tool iteratively and randomly picks interfaces in the program and merges them until all merged interfaces are implemented by at least t classes or until it can no longer find interfaces to merge within a jar. The latter occurs a lot for `eclipse`, of which the classes are partitioned over many more jar files. For each merge threshold value, the tool generates ten different program versions with ten different random seeds. When we present results for a level of merging in later charts, we always present the average result obtained for the ten versions at that level. In the charts, a merging threshold of 0 refers to the basic CHF algorithm without interface merging. In our proof-of-concept tool, the dummy method bodies are empty. Other extensions as described in Section 3.2 are left for future work. All generated program versions were type verified and proved to work correctly on the DaCapo inputs.

4.2 Results

Protection against Human Program Understanding. As all obfuscation researchers, we face the problem of measuring the potency of our technique. And as in almost all of the literature (see, e.g., the literature discussed in Section 5), we know of no suitable metrics that directly measure the resistance to, e.g., reverse-engineering attacks. Therefore we instead rely on established software complexity metrics from the domain of software-engineering. In particular, we use the static QMOOD metrics from Bansiya et al. [1]. QMOOD stands for Quality Model for Object-Oriented Design. It includes a metric for understandability that is defined as a linear combination of other complexity metrics that measure different aspects of a design, including abstraction, encapsulation, coupling, cohesion, polymorphism, complexity and size [1]. This understandability metric is a relative metric that can only be used to compare two program versions. Given an original program with a normalized understandability score of -0.99 (as defined in [1]), less understandable versions will have lower scores. Figure 7(a) displays the relative understandability for the four benchmark programs. CHF clearly reduces human understandability significantly, with understandability dropping as more interfaces are merged. For `eclipse`, less interfaces got merged at higher merge thresholds because its classes are partitioned over more jar files. This results in a higher understandability than the other benchmarks.



(a) understandability results



(b) QMOOD breakdown for `batik`

Fig. 7. QMOOD understandability

Figure 7(b) shows the breakdown of `batik`'s understandability over its components for the six threshold levels. For other benchmarks, similar results are obtained. Each bar depicts the relative value of one metric compared to the value of that metric of the baseline program. It can be observed that CHF influences abstraction, coupling, cohesion, and complexity of the program, of its classes and of its class hierarchy. As such, the impact of CHF on the effort needed by an attacker to reverse-engineer and understand the program is multidimensional.

For all benchmarks, the variation in understandability score among the 10 program versions generated for each level of interface merging was at most 12%, the large majority of which was below 9%. This shows that interfaces can be merged in less or more confusing combinations, but also that the decrease in understandability is determined more by the level of merging than by the particular combinations merged.

Protection against Static Analysis Tools. We measure the ability to confuse static analyses in terms of increases in points-to set sizes. In practice, the precision of many important client analyses, including call graph construction [14] and virtual call resolution, can drop significantly as the result of an imprecise points-to analysis.

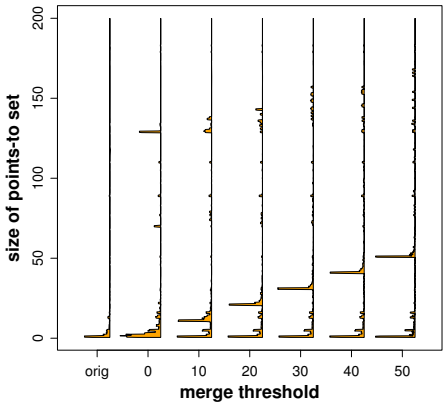


Fig. 8. Points-to set sizes in `batik`

At the same time, the analysis costs such as memory footprint and execution time increase with less precise points-to analyses because the constructed call graphs becomes bigger. Hence, reducing the precision of points-to analyses by causing them to return larger points-to sets, will directly reduce both the effectiveness and the efficiency of several static analyses that are fundamental for static code attacks. We made Soot compute the points-to sets with class hierarchy analysis [10], the points-to analysis that cannot be hampered by

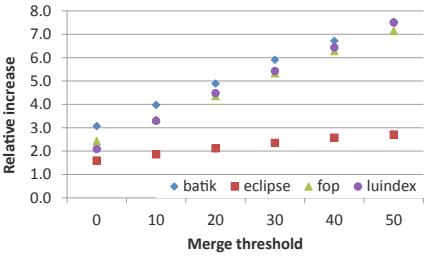


Fig. 9. Code size overhead

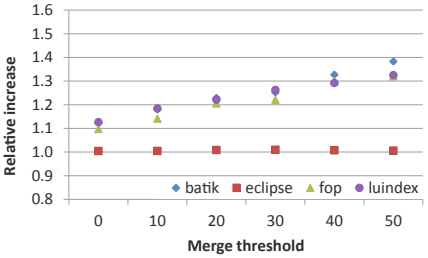


Fig. 10. Memory footprint overhead

object allocation obfuscation as discussed in Section 3.2. The histogram in Figure 8 depicts the distributions of points-to set sizes of all local variables and parameters in the methods of classes declared as (transformable or not) application types in the `batik` benchmark. For the other benchmarks, we observed very similar trends.

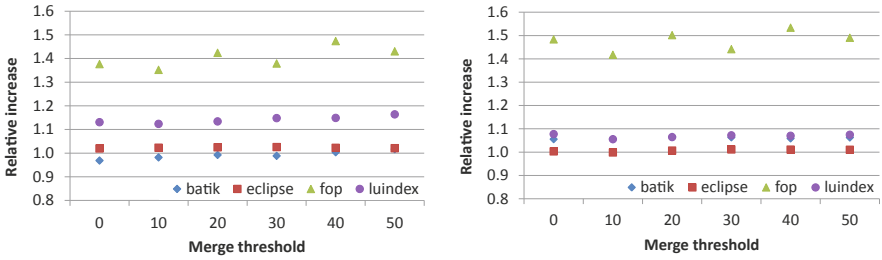
Clearly CHF increases the sizes of many points-to sets. In particular those points-to sets of variables declared as merged interface types grow with the number of classes implementing those interfaces. In all benchmarks, a considerable number of points-to sets does not grow even when more interfaces are merged. This follows from the fact that the increases are limited to the points-to sets of variables whose type is changed during CHF.

Not visible in the histograms, but similarly to what we observed for QMOOD, the points-to set size increases depend much more on merging threshold than on the particular combinations of interfaces merged.

Overhead. Figure 9 depicts the relative code size increase as a result of the basic flattening and interface merging. Overall, more interface merging implies more code. The increase, which results mainly from methods being duplicated and (mostly) empty dummy methods being added, varies from one benchmark to the other. The lower increase for `eclipse` is caused by its classes being partitioned over more jar files, as a result of which fewer interfaces got merged.

Figure 10 depicts the relative memory footprint increase observed with the Java SE Runtime Environment (build 1.6.0_30-b12) and the Java HotSpot 64-bit Server VM (build 20.6-b03) on standard runs consisting of 10 consecutive program executions in a benchmark harness on the default inputs. In general, the memory footprint overhead is a linear function of the code size overhead because classes and code are also stored in memory. The memory footprint overhead is an order of magnitude smaller, however, because there are many class instances (i.e, objects) allocated on the heap whose size is unaffected by CHF.

Finally, Figure 11 depicts the performance overhead in terms of the relative execution time increase. The overheads reported in Figure 11(a) include all 10 runs of the benchmarks in their harness. This includes the warm-up runs during which the JIT compiler is very active. As the code size grows with interface merging, so does the time spent by the JIT compiler. Figure 11(b) shows that during steady-state (i.e., when only the last of the 10 runs is considered during



(a) warm-up included (b) steady-state

Fig. 11. Performance overhead

which almost no JIT compilation takes place anymore) the performance overhead no longer depends on the merging threshold value. For most benchmarks, the overhead is limited to less than 10%. For `eclipse`, the overhead is insignificant because most of its execution time is spent in non-transformable classes. The small variations in the observed execution time of `eclipse` are clearly within the expected noise range [13]. For `fop`, considerably more overhead remains. A performance analysis revealed that the getters and setters introduced during the interface insertion are the culprits.

5 Related Work

To obfuscate the overall application design, and in particular its class hierarchy and the type information contained in the code, Sosonkin et al. proposed class coalescing, class splitting, and type hiding by introducing interface types and by replacing declarations of class types with declarations of those interfaces [28]. In its most extreme form, their class coalescing transformation can coalesce all transformable classes in the program into a single class, effectively removing the whole program design; beyond what CHF can achieve. For example, when all classes are coalesced, all points-to sets becomes singletons that contain all types in the program. In other words, points-to sets become completely useless. The main disadvantage of class coalescing is that the number of member fields in coalesced classes grows far beyond the number of original member fields in the original classes and all their superclasses. As a result, their instances also grow bigger, which will lead to a much larger memory footprint. The authors acknowledge this potential issue, but their experimental evaluation is limited to execution time measurements of relatively small programs (up to 307 classes). For those, they measure slow-downs up to 130% even with limited coalescing. Furthermore, their evaluation does not contain any criteria related to software protection, software understandability, or software complexity, and when limitations to the application of their transformations are observed, they hide behind tool maturity instead of investigating more fundamental issues. By contrast, this paper proposed an obfuscation that from the very start maximally

removes the class hierarchy, and of which the code size, memory footprint, and (smaller) performance overhead are evaluated in detail, as well as its impact on program understandability, for a set of large, real-life programs (up to 5947 classes). Furthermore, rather than being immature, our prototype tool pushes the application of our obfuscation to the fundamental limits relating to external libraries, dynamic class loading and reflection.

The false factoring transformation proposed by Collberg et al. [8] refactors a program in such a way that two or more unrelated classes come to share a superclass, thereby giving the impression that they are related. We know of no public tool implementing this proposal or of any experimental evaluation of it.

Given a set of transformable classes, the obfuscation techniques introduced by Sakabe et al. [24] first changes the signature of all methods in the classes such that each class implements the same set of overloaded methods. These methods are then defined in an interface implemented by the classes and used in declarations instead of the original classes. To hide the actual type of objects bound to variables of the interface type, they propose to replace single object creations by a set of object creations guarded by opaque predicates. CHF as presented here is to a certain degree complementary, as explained in Section 3.2.

In the field of software refactoring, Snelting and Tip [26,27] presented a method for analyzing and reengineering class hierarchies by extracting information on the use of an application's class hierarchy, from which they construct a concept lattice that provides insights on how to improve the hierarchy to better match the way the classes interact. Their analysis can detect where class members can be moved to a subclass or identify where it is beneficial to split classes. This analysis has been extended and implemented in the refactoring tool KABA [25,29,30]. This tool uses the results from the concept analysis to present several refactorings to the user, who can then interactively modify the class hierarchy. Potentially, Snelting and Tip's work could help an attacker find related classes in a flattened hierarchy by allowing him to see through the smokescreen of specially crafted dummy method implementations and by detecting unrelated classes implementing merged interfaces. It remains an open question to assess to which extent their tool would be useful in practice.

6 Conclusions and Future Work

This paper presented class hierarchy flattening, an obfuscating program transformation for object-oriented programs written in managed code languages. The transformation removes the class hierarchy to the extent possible to hide the overall application design. We presented the basic technique and possible extensions. Together with the basic algorithm, one of those extensions, called interface merging, was evaluated extensively on large real-world programs. While several aspects of the experimental results deserve further analysis, they clearly demonstrate that class hierarchy flattening provides measurable protection against at least some forms of human understandability and automated program analysis. This protection is achieved at relatively low levels of run-time overhead.

Our future work will concentrate on more focused interface merging strategies to outperform random merging, on extending the basic protection as discussed in the paper, and on a more extensive evaluation involving more security metrics, including diffing-based metrics, and more complex points-to analyses.

References

1. Bansiya, J., Davis, C.G.: A hierarchical model for object-oriented design quality assessment. IEEE Trans. Softw. Eng. 28(1), 4–17 (2002)
2. Batchelder, M., Hendren, L.: Obfuscating Java: The Most Pain for the Least Gain. In: Adsul, B., Odersky, M. (eds.) CC 2007. LNCS, vol. 4420, pp. 96–110. Springer, Heidelberg (2007)
3. Blackburn, S.M., McKinley, K.S., et al.: Wake up and smell the coffee: evaluation methodology for the 21st century. Commun. ACM 51(8), 83–89 (2008)
4. Bodden, E., Sewe, A., et al.: Taming reflection: Aiding static analysis in the presence of reflection and custom class loaders. In: Proc. ICSE, pp. 241–250 (2011)
5. Chan, J.T., Yang, W.: Advanced obfuscation techniques for Java bytecode. Journal of Systems and Software 71(1-2), 1–10 (2004)
6. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: White-Box Cryptography and an AES Implementation. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 250–270. Springer, Heidelberg (2003)
7. Collberg, C., Nagra, J.: Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection. Addison-Wesley Professional (2009)
8. Collberg, C., Thomborson, C., Douglas, L.: A taxonomy of obfuscating transformations. Technical report, University of Auckland (1997)
9. Collberg, C., Thomborson, C., Low, D.: Manufacturing cheap, resilient, and stealthy opaque constructs. In: Proc. ACM POP, pp. 184–196 (1998)
10. Dean, J., Grove, D., Chambers, C.: Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis. In: Olthoff, W. (ed.) ECOOP 1995. LNCS, vol. 952, pp. 77–101. Springer, Heidelberg (1995)
11. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston (1999)
12. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994)
13. Georges, A., Buytaert, D., Eeckhout, L.: Statistically rigorous java performance evaluation. In: Proc. ACM OOPSLA, pp. 57–76 (2007)
14. Grove, D., Chambers, C.: A framework for call graph construction algorithms. ACM Trans. Program. Lang. Syst. 23(6), 685–746 (2001)
15. Hind, M., Pioli, A.: Evaluating the effectiveness of pointer alias analyses. Science of Comp. Programming 39(1), 31–55 (2001)
16. Holst, S.: Assessing and managing security risks unique to Java and .NET. ISSA Journal (2009)
17. Hou, T., Chen, H., Tsai, M.: Three control flow obfuscation methods for Java software. IEE Proceedings-Software 153(2), 80–86 (2006)
18. Lam, P., Bodden, E., Lhoták, O., Hendren, L.: The Soot framework for Java program analysis: a retrospective. In: Proc. CETUS 2011 (October 2011)
19. Majumdar, A., Thomborson, C.D.: Manufacturing opaque predicates in distributed systems for code obfuscation. In: Proc. ACSC, pp. 187–196 (2006)

20. Majumdar, A., Thomborson, C.D., Drape, S.: A survey of control-flow obfuscations. In: Bagchi, A., Atluri, V. (eds.) ICISS 2006. LNCS, vol. 4332, pp. 353–356. Springer, Heidelberg (2006)
21. McCluskey, E.: Introduction to the theory of switching circuits. McGraw Hill Text (1965)
22. Palsberg, J., Krishnaswamy, S., Kwon, M., Ma, D., Shao, Q., Zhang, Y.: Experience with software watermarking. In: Proc. ACSAC, pp. 308–316 (2000)
23. Ryder, B.G.: Dimensions of Precision in Reference Analysis of Object-Oriented Programming Languages. In: Hedin, G. (ed.) CC 2003. LNCS, vol. 2622, pp. 126–137. Springer, Heidelberg (2003)
24. Sakabe, Y., Soshi, M., Miyaji, A.: Java obfuscation approaches to construct tamper-resistant object-oriented programs. IPSJ Digital Courier 1, 349–361 (2005)
25. Snelting, G., Streckenbach, M.: KABA: Automated refactoring for improved cohesion. In: Proc. of the First Workshop on Refactoring Tools, pp. 1–2 (2007)
26. Snelting, G., Tip, F.: Reengineering class hierarchies using concept analysis. In: Proc. ACM FSE, pp. 99–110 (1998)
27. Snelting, G., Tip, F.: Understanding class hierarchies using concept analysis. ACM Trans. Program. Lang. Syst. 22(3), 540–582 (2000)
28. Sosonkin, M., Naumovich, G., Memon, N.: Obfuscation of design intent in object-oriented applications. In: Proc. ACM DRM, pp. 142–153 (2003)
29. Streckenbach, M.: KABA - a system for refactoring Java programs. PhD thesis, Universität Passau (2005)
30. Streckenbach, M., Snelting, G.: Refactoring class hierarchies with KABA. In: Proc. ACM OOPLSA, pp. 315–330 (2004)
31. Tip, F., Furher, R., Kieżun, A., Ernst, M., Balaban, I., De Sutter, B.: Refactoring using type constraints. ACM Trans. Program. Lang. Syst. 33(3), 9:1–9:47 (2011)
32. Vallée-Rai, R., Co, P., Gagnon, E., Hendren, L., Lam, P., Sundaresan, V.: Soot - a java bytecode optimization framework. In: Proc. CASCON, pp. 125–135 (1999)
33. Venkatraj, A.P.R.: Program obfuscation. Master's thesis, University of Arizona (2003)
34. Zhou, Y., Main, A., Gu, Y.X., Johnson, H.: Information Hiding in Software with Mixed Boolean-Arithmetic Transforms. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 61–75. Springer, Heidelberg (2008)

RESource: A Framework for Online Matching of Assembly with Open Source Code

Ashkan Rahimian[1], Philippe Charland[2], Stere Preda[1], and Mourad Debbabi[1]

[1] Computer Security Laboratory, CIISE
Concordia University, Montreal, Quebec, Canada
{a_rahimi,s_preda,debbabi}@encs.concordia.ca
[2] Mission Critical Cyber Security Section,
Defence R&D Canada - Valcartier, Quebec, Canada
philippe.charland@drdc-rddc.gc.ca

Abstract. Software reverse engineering is a fastidious task demanding a strong expertise in assembly coding. Various existing tools may help analyze the functionality of a binary file without executing it and an interesting step would naturally be the search for the original source files. Our tool called RESource considers the extraction of some features in the assembly code so that queries can be triggered to a source repository in a reliable way: either (1) the result is a set of references to the original project files provided they are hosted on the repository or (2) at least some functionalities of the binary file are unleashed. Such an approach is very promising given its proved performances in real assembly code applications.

Keywords: reverse engineering, assembly code, source repository.

1 Introduction

Software reverse engineering consists in studying and understanding the process by which a machine-generated assembly language program has been created by working backward [3]. If manually writing assembly ASM code involves specific programming skills, a compiler automatically converts a high-level language such as C into machine code. The ASM analysis becomes extremely challenging, especially if the compiler adds certain optimizations by rearranging the computations, changing or replacing some operations.

Common reverse engineering practices suggest two approaches – dynamic and static – with the binary file as the starting point. By dynamic approach (e.g., [11]) we mean isolating the binary file in an application specific environment to model its behavior by execution. Since this does not necessarily reveal all execution flows, debugging tools (e.g., WinDbg [18], Gdb [5], Valgrind [17]) are often associated with this method. As long as only the functionality is targeted, the dynamic approach is acceptable. In other situations, static analysis yields better results and does not compromise the security requirements of the analysis environment.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 211–226, 2013.

The first step of the static analysis of a binary file is the disassembly phase. The disassembler (e.g., objdump [6] in Linux) is a program considered of invaluable help since it generates the ASM code of the binary file. At this level, mastering the ASM program representation seldom leads to fully understanding the program functionalities. More advanced disassemblers such as IDA Pro [9] are meant to simplify the analysis by offering a rich GUI with the program divided into basic blocks in a program flow graph (PFG). A challenging further step is then to obtain a correct higher-level program representation, i.e., the source files. A decompiler (e.g., Hex-rays [14] or TyDec [16]) could help a lot but since there is not always a 1:1 correlation between the ASM and the source objects, the automatically generated sources may be difficult to follow. For example, it is not simple to detect the definition of object structures in ASM.

Our current purpose is a tool – RESource – to help enforce the mapping between the machine and the source code. For that we draw inspiration from the RE-Google project [10]. RE-Google was designed on top of the GData framework and Google Code Search APIs [7] which were officially deprecated. Our tool provides a functionality similar to RE-Google and introduces new ones. The approach is the following: with the principle of code reuse in mind, our tool will exploit some features that exist at both the source and the assembly file levels. The tool is thus able to trigger queries based on these features on certain repositories used by the developers' community. If there are no query results, the tool is still able to give us some information about the functionality of a portion of the ASM file using an Offline Analyzer module. The information returned by this module are related to the function stack frame, prototype, arguments, local variables and low-level system calls. It has a built-in dictionary of common user and kernel level API functions that are used by malware to interact with the Windows operating system for performing tasks such as file I/O, network communications, registry modification, working with services, etc.

The reminder of the paper is structured as follows: first we shall introduce the motivation of our work and related background. In Section 3 we present our methodology followed by implementation details. Two experimental scenarios are described in Section 4, followed by the conclusion in Section 5.

2 Motivation and Related Work

The current work pertains to the domain of static analysis of ASM code and more precisely, in the mapping of ASM to source files. Though a decompiler seems to be the program which best fits our goals, we consider that an attempt of mapping the ASM to existing source code should come first. Applications such as malware analysis can grasp the benefits of a tool able to give reliable information about the standard and open source files used by a malicious developer. Decompilers, methods and tools to analyze malware code already exist. They can be used by expert reverse engineers who seek to understand the origins and the creation process of the malware.

IDA Pro allows disassembling a binary file and its rich GUI simplifies the analysis of the ASM code. It is widely used thanks to its multiple features, such as the possibility of integrating a debugger like WinDbg (which became the de facto Microsoft debugger since Softice stopped being maintained) and more interesting, plugins such as the Hex-Rays decompiler - "the most advanced decompiler ever built!" [14]. Although there have been a few attempts to design generic debuggers to work on heterogeneous platforms (e.g., GenDbg [4]), IDA Pro proves to be one of the most complete tool in reverse engineering.

The ASM code follows rather regular patterns. Consequently, the decompiler is able to do a mapping between registers or memory locations, abstract variables, and thus extract for example, a C-like program from the ASM file (indeed, most of the decompilers are not generic). Other basic C constructs (e.g., loops) are more difficult to extract and some decompilers fail to solve them (e.g., Boomerang [2]). Another challenging problem is reconstructing the abstract types (e.g., structures). TyDec [15] tries to tackle the problem but is limited to an experimental level. In this case, the best practice remains the human expertise, i.e., the definition of a structure which is guessed after a first decompilation is manually introduced and the C program is then rewritten.

Our approach is rather different in that our RESource tool is meant to inform the reverse engineer about the standard and open source components that might have been used by the creator of the binary file. To our knowledge, a similar functionality is ensured only by the IDA Pro RE-Google plugin [10].

RE-Google, written in Python, relies on the IDA API and the Google Code Search API [7]. It takes the disassembled binary code as input and creates a query submitted to Google Code Search based on the constants, strings, and function names. The response from the search engine is the potential source excerpt that contains similar code. Although it supports a limited set of features to create a query, RE-Google may confine the search to certain languages. Additionally, it can be configured to search for a specific function within the disassembly, skip certain functions, or perform a search for all available functions. Also, the interval between two subsequent searches can be defined. Optionally, user credentials could be supplied as part of the query to the code search engine. Furthermore, there is an option for restricting (blacklisting) certain string patterns in the result. Similarly, a constant filter function checks the immediate values and removes flags and small values form the query if they are not relevant for the search. The response from the search engine is parsed and the top results are added to the code as comments.

Our goal is to design RESource as an IDA Pro plugin too, making use of code search engines for open source software. In addition, we want to have the capability to search in newsgroups and user-defined code repositories, taking thus into account a larger panel of search engines than RE-Google. RESource does not only provide extended queries by adding new features. It also allows to reveal parts of the code functionalities whenever the query results are null. In the next section, we describe the methodology and the concrete implementation details of our RESource tool.

3 Methodology

The input to our process is the ASM file resulting from the disassembly of a target binary in IDA Pro. The specific representation of the ASM, together with its PFG, lead us to consider the partitioning of the ASM code in blocks, each one corresponding to begin *proc / end proc*, where proc stands for procedure. Here is an example of a simple code in C.

```
int sum(int a, int b){
     return a + b;
}
```

The corresponding ASM code contains a procedure that we can easily identify by its *name* "sum" (IDA Pro encloses it with the *begin proc* and *end proc* keywords). **sum** :

```
push   %ebp
mov    %esp,%ebp
mov    0xc(%ebp),%eax
add    0x8(%ebp),%eax
pop    %ebp
ret
```

We thus consider an ASM file as a set of procedures that are to be individually analyzed by our tool. Each procedure may contain some *interesting features* (see Section 3.2) that our tool is able to extract and exploit in order to submit queries to a source repository. The result is (1) either a set of links to pertaining source files referencing the same features and which are inserted as comments in the original ASM file or (2) the insertion of a comment about the functionality of the current procedure after its local offline analysis (cf. Section 3.2).

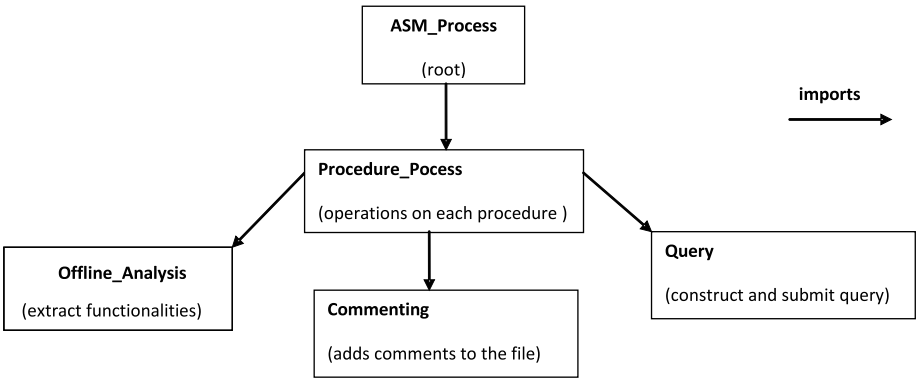


Fig. 1. Algorithm Decomposition

3.1 Algorithm

We adopt a B-Method like notation [1] to describe the algorithm implemented by our RESource tool. Fig. 1 captures the RESource algorithm decomposition in B-like components. The algorithm has five modules: (1) *ASM_Process* root machine which provides the interface with the user. It imports the (2) *Procedure_Process* machine responsible for processing each ASM procedure. It calls the operations of the (3) *Query* module in order to submit queries to a *set* of code repositories. The (4) *Offline_Analysis* module is in charge of a local analysis to extract the program functionality and also the operations of the (5) *Commenting* module which adds the pertaining comments to the original file.

***ASM_Process* Module**

```
MACHINE ASM_Process
IMPORTS Procedure_Process
SETS
    PROCEDURES
CONSTANTS
    ASM_Original_file
PROPERTIES
    ASM_Original_file ∈ ℙ(PROCEDURES)
VARIABLES
    Some_Procedures
INVARIANTS
    Some_Procedures ⊆ ASM_Original_file
INITIALISATION
     Some_Procedures := ∅
OPERATIONS
    try_read_procedures(procs) = PRE procs ≠ ∅ ∧ procs ⊆ ASM_Original_file
                                        THEN Some_Procedures := procs
                                    END;
     process = PRE Some_Procedures ≠ ∅ THEN
                 VAR F1, F2, p IN
                    WHILE Some_Procedures ≠ ∅ DO
                        ANY p WHERE p ∈ Some_Procedures THEN
                          F1, F2 ← read_features(p); /*from Procedure_Process*/
                          query(F1); /*op. in Procedure_Process*/
                          analyse_locally(F2); /*op. in Procedure_Process*/
                          update(p); /*op. in Procedure_Process*/
                          Some_Procedures := Some_Procedures \ {p};
                        END
                    END
                 END
                END
END/*ASM_Process*/
```

Fig. 2. ASM_Process module

Any ASM file is a SET of PROCEDURES. As we can easily depict from Fig. 2, we take as input to our algorithm the ASM_Original_file. It is of type PROCEDURES and remains CONSTANT. These assumptions are captured by the CONSTANT and PROPERTIES clauses. The only variable we introduce is a set of SOME_PROCEDURES among those presented by IDA Pro that the user chooses to analyze. This variable may be modified by the OPERATIONS which must always satisfy the INVARIANT. Here, the INVARIANT states that the procedures to be analyzed are part of the original ASM file.

Procedure_Process **Module**

MACHINE *Procedure_Process*
IMPORTS *Query, Offline_Analysis, Commenting*
SETS
FEATURES
VARIABLES
OnFeat, OffFeat, queried, analysed, updated
INVARIANTS
OnFeat ⊆ FEATURES ∧ OffFeat ⊆ FEATURES ∧ queried ∈ **BOOL** ∧
analysed ∈ **BOOL** ∧ updated ∈ **BOOL**
INITIALISATION
OnFeat, OffFeat, queried, analysed, updated := ∅, ∅, false, false, false
OPERATIONS
F1, F2 ← *read_features*(p) = **PRE** p ≠ ∅ **THEN**
`/*features extraction from p: to refine*/`
F1:=OnFeat; `/*features for online analysis*/`
F2:=OffFeat; `/*features for local analysis*/`
queried, analysed, updated := false, false, false;
END;
query(f) = **PRE** f ⊆ OnFeat ∧ queried = false **THEN**
IF f ≠ ∅ **THEN**
submit_query(f); `/*operation in Query machine*/`
END
queried := true;
END;
analyse_locally(f) = **PRE** f ⊆ OffFeat ∧ queried = true ∧ analysed = false
THEN `/*local analysis for functionality extraction */`
`/*based on operations in Offline_Analysis machine*/`
analysed := true; append_to_log(f); `/*displaying results*/`
END;
update(p) = **PRE** queried = true ∧ analysed = true ∧ updated = false **THEN**
`/*updates after the online query and the local analysis*/`
`/*based on operations in Commenting machine*/`
updated := true;
END;
END`/*Procedure_Process*/`

Fig. 3. Procedure_Process module

For each procedure, there is a phase of ASM *features* extraction, followed by the submission of queries to source repositories and a local analysis.

We explain these steps in the *process* operation of Fig. 2. The Procedure_Process module uses respectively the services of the Query and the Offline_Analysis modules for the specific *query* and *analyse_locally* operations (Fig. 3). These operations are to be carefully implemented since their abstract representation cannot contain too many details. The module states only the permitted order in which these operations are called via the PRE-condition clause.

The *process* operation of Fig. 2 considers a last phase of *updating*. The original ASM file remains the same, i.e., *constant*, except for the ASM comments part which gathers the query results and the local analysis. Therefore, the result of the entire process is the original file updated with comments as we shall see in Section 3.2.

Query Module

MACHINE *Query*
CONSTANTS
n, SEQ_REPS
DEFINITIONS
Repositories == 1..n
PROPERTIES
n ∈ ℕAT1 ∧ SEQ_REPS ∈ Repositories → Repositories
VARIABLES
Queryable_Reps
INVARIANTS
Queryable_Reps ∈ Repositories → **BOOL**
INITIALISATION
ran(Queryables_Reps) := true
`/* all repositories should be queryable at the beginning*/`
OPERATIONS
submit_query(F) = **ANY** r **WHERE** Queryable_Reps(r) = true **THEN**
`/*submit query*/`
Queryable_Reps(SEQ_REPS(r)) := true;
END
END`/*Query*/`

Fig. 4. Query module

Based on the extracted features in a procedure, the role of the Query module is to construct and submit queries to a set of source repositories which are previously known. We could use an instantiated SET of repositories to capture this information, but for the sake of simplicity, we choose to identify each source repository with a natural number in the set 1..n, where n is the number of repositories.

Moreover, we also express the following requirement: a real source repository may not be queried too frequently (e.g., wait a few seconds between each query). Consequently there should be a mechanism to launch the query to a different

queryable repository so that the process does not stop. The straightforward way is to introduce a CONSTANT function SEQ_REPS which gives the *next* source repository to query. Implementing this is based on the observation of some query interval slots for each real repository and by thus defining an order of passing from one repository to another. Queryable_Reps($\mathbb{r}$) = true therefore means that the $\mathbb{r}$ repository can accept a query. This variable is modified in the implementation of the submit_query operation.

We do not give the B notation of the Offline_Analysis and Commenting modules because their operations proved to be more challenging to implement at low level. The *append_to_log*() operation is meant to save the execution steps in a log file at runtime.

3.2 Implementation Details

If a B-like algorithm description is useful to examine the possible flows and to define the operations preconditions and the invariants they have to meet, the validity of the low level implementation is generally asserted using normal techniques such as testing and peer code reviewing.

RESource program implements the algorithm as a Python IDA Pro plug-in. It is worth mentioning that, unlike the RE-Google plugin [10], our extended version does not rely on the GData framework [8], nor does it utilize Google Code [7] as the only search engine for accessing code repositories. Instead, it possesses a built-in query processing engine and parsing mechanism for handling request/response messages. Furthermore, it supports multiple search engines and it provides a framework for adding new code repositories with only a few lines of code. Also, the program makes use of an interleaving time optimization technique for managing multiple search engines. Despite the large number of request/response messages, it honors the required time delays between consequent messages without wasting processing time.

In terms of extracted *interesting features* from the ASM code, RESource is able to get four types of features for online analysis and query building: (1) immediate values of operands, (2) imported libraries and function calls, (3) exported functions in DLLs, and (4) strings values. In addition, it considers eight features for offline analysis. For each function, we extract information about its stack frame: (1) number of instructions; (2) size and number of local variables; (3) size and number of arguments; (4) size of saved registers; (5) function flags; (6) function addresses (begin, end, return); (7) function prototype (type of input and output and calling convention); (8) calls to low level system functions (malware dictionary). Moreover, variable scopes (local/stack-based or global/memory-based) and simple data structures (single variables or *structs*) are also highlighted for the reverse engineer.

Moreover, the program adds better result handling techniques than RE-Google and an offline functionality analysis engine. In many situations, online results may not be available due to the lack of extracted features, obfuscated or hard-coded procedure, use of complex and non-standard algorithms, etc. Therefore, the offline analyzer is of great benefit for revealing the overall functionality of a

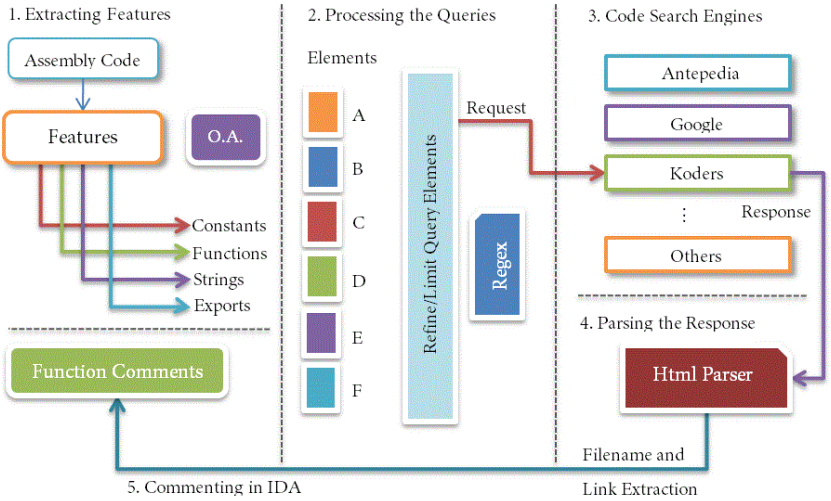


Fig. 5. Execution Flow

portion of assembly code. It has an extendable dictionary of common functions in Windows API along with a programmer-friendly description of each function.

Execution Flow

As illustrated in Fig. 5, there are five main modules in the Python program for handling tasks related to Features, Queries, Repositories, Parsing and Commenting. Except for the Code Search Engine (3), these modules have a counterpart in the algorithm blocks of Fig. 1. The RESource program interacts with the IDA Pro API for getting a list of available procedures in the disassembly, getting function addresses and names, as well as adding comments to the file.

The execution flow starts in the main function of the script where the initialization of variables and execution time calculation is done (*Initialize*(*RESrc_Vars*)). Then, the script checks a variable (flag) to determine whether the search should be performed on a specific function or on all the extracted functions from the disassembly (*RESrc*(*asm_function_list*)). In the first case, the user highlights a specific function for search and in the second case, all the functions are taken into account.

In the next step, the RESrc function counts the total number of available procedures and prepares a loop for analyzing each item. Then, a function will be called for extracting four types of features, namely constants, imported libraries, exported libraries and string values from the disassembly. The output of the Extract_QFeatures() function is a potential list of features that could be used for building a general query. This list will be refined several times before building a specific query. Next, the features for offline analysis are extracted using the Extract_OAFeatures() function :

```
feature_qlist ← Extract_QFeatures( asm_function )
feature_oalist ← Extract_OAFeatures( asm_function)
            // OA for Offline Analysis
    feature_list_func_i = [ f_i1, f_i2, ..., f_in ]
refined_qlist ← Refine_GQuery( feature_qlist )
```

The purpose of the *Offline Analysis* (OA) module is to compare a function with a list of known Windows API functions in order to get a simple statement about the functionality and prototype of the procedure under analysis. Also, this module assists the reverse engineer by highlighting the variables and their scope.

The purpose of the Refine_GQuery() function in the refining process is to filter out certain characters from the feature list to prevent problems with search engines queries. For instance, the search engines may not allow characters such as "%, ', ' "' as part of query string to prevent SQL injection. Therefore, the output query is safe for submission into code search engines. However, the user can define what characters are blacklisted by adding 'badkey': 'value' pairs into the "BlackDict" dictionary. For instance, the keys in the following dictionary are simply replaced with the '' character which is equivalent to removing them from the search string.

```
BlackDict = {'%d':'', '%s':'', '\\':'', '%1':'', '%2':'', ...}
```

Also, this function encodes and prepares the list for the next steps of specific query building:

```
base_query ← Generate_Query( refined_qlist )
```

At this step, we have a base query that can be further encoded for particular search engines. Search engine-specific prefix and suffix will be added to each base query to build a standard query. The following functions are examples of query building functions for three major code search engines.

```
final_queryKoders ← Build_Koders_Query( base_query )
   final_queryGCS ← Build_GCS_Query( base_query )
final_queryKrugle ← Build_Krugle_Query( base_query )
```

The next step is to submit the query and get the response for each respective search engine. The order of query submission and response extraction is important for time optimization. Usually there must be a time delay between two subsequent requests to a search engine (SE). The program uses an interleaving technique for managing the query submission and for saving processing time (as shown for example in Fig. 6).

For each query, a request is made and the response page is received in HTML.

```
html_page_j ← Fetch_Response( final_query_j )
```

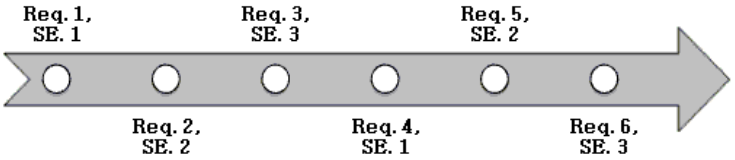


Fig. 6. Timing Interleaving

After getting the page, a call to the local parsing function will be made to extract relevant information based on a predefined regular expression statement for each search engine. Then, the matching Filenames and URLs are extracted and stored in a dictionary.

```
dictionary_list ← Refine_Results( Parse_Page( html_page ) )
```

The results are processed and duplicate results are removed from the list. Also, based on the search engine rankings, the best matches are selected and given a priority. Lastly, the comments are updated to reflect the online search results.

$\texttt{function\_comment} \leftarrow \texttt{Update\_Comment}_{function\_k}\texttt{(refined\_dictionary\_list)}$

In the next section, we present a practical application of our RESource program on an open source software.

4 Experimental Results

We have adopted the PreciseCalc Project [12] given that both the sources and binary files are available on SourceForge and Koders (*http://www.koders.com*) as a code search engine. As an input to our RESource IDA Pro plugin, we use the ASM file resulting from disassembling the PreciseCalc binary. There are 533 ASM procedures and we choose to analyse all of them.

The `Extract_QFeatures()` function is able to extract features from 67 procedures. If there are at least two elements in the *Imports list* or the joint set of *Constants* and *String List* is non-empty, then the script would try to find an *exact match* by concatenating all the elements. This is an ideal situation where the query would be expressive enough in terms of number and the type of elements. If no exact match is found, then the search would be based on the strings inside the binary. If the length of *String List* is larger than one, then the search query will be built by concatenating the String elements. Finally, if there is at least one element in the *Constants List* but the results set is empty, the script will perform the search by building a query based on the concatenation of constant elements.

The conditional rules for defining each case can be altered based on the application under analysis and the number of available elements in the extracted lists. Generally, there are more elements in each list when the application makes use of Standard Windows Libraries.

1	* Analyzing function 4 [sub_4013A0] @ [0x4013a0] Constants: ['0x13880', '0xf4240', '0x4c4b40', '0x493e0'] Strings: [] Imports: set([]) list: 0x13880 0xf4240 0x4c4b40 0x493e0 * Looking for an exact match * * Looking for a rough match based on constants *
2	* Analyzing function 127 [sub_407F40] @ [0x407f40] Constants: [] Strings: ['"arctan, arccot from 1i or -1i"', '"argtanh,argcoth from 1 or -1"'] Imports: set([]) list: "arctan, arccot from 1i or -1i" "argtanh,argcoth from 1 or -1" * Looking for an exact match * * Looking for a close match based on strings *
3	* Analyzing function 334 [sub_417B20] @ [0x417b20] Constants: ['0x80000001', '0x80000001', '0x80000001'] Strings: ['"Software\\Petr Lastovicka"', '"Software\\Petr Lastovicka"'] Imports: set(['RegDeleteKey', 'RegOpenKey', 'RegCloseKey', 'RegQueryInfoKey']) list: 0x80000001 0x80000001 0x80000001 "Software\Petr Lastovicka" "Software\Petr Lastovicka" RegDeleteKey RegOpenKey RegCloseKey RegQueryInfoKey * Looking for an exact match * * Looking for a close match based on strings * * Looking for a rough match based on constants *
4	* Analyzing function 375 [sub_41B4A0] @ [0x41b4a0] Constants: [] Strings: ['"sin"', '"cos"', '"tan"', '"tg"', '"cot"', '"cotg"', '"asin"', '"acos"', '"atan"', '"atg"', '"acot"', '"acotg"', '"arcsin"', '"arccos"', '"arctan"', '"arctg"', '"arccot"', '"arccotg"', '"arc"'] Imports: set(['strnicmp']) list: "sin" "cos" "tan" "tg" "cot" "cotg" "asin" "acos" "atan" "atg" "acot" "acotg" "arcsin" "arccos" "arctan" "arctg" "arccot" "arccotg" "arc" strnicmp * Looking for an exact match * * Looking for a close match based on strings *

Fig. 7. Feature Extraction (excerpt from the *log* file)

Fig. 7 shows a few examples of interesting features extracted by the `Extract_QFeatures()` function. In the first one, the search is merely performed based on the *constants*. Example 2 shows a situation in which only *string* information is available. No import lists are detected for the first two cases. Conversely, in examples 3 and 4, sets of *imported function* names are included in the search. The original PreciseCalc project can be accurately identified by submitting a query containing portions of the strings in example 3. Even if an exact match is not found, the RESource program will try to find a close or a rough match based on a combination of features.

RESource was able to detect several references to each source file in the original project. PreciseCalc application includes functions that handle Text Editing, GUI Processing, Timing and Registry Modification, alongside the Arithmetic, Statistical, Geometrical and other math-related functions.

Table 1 shows sample results of the identified C++ source code. The identified links and filenames are inserted directly in the assembly file. There are several references to the main "preccalc.cpp" file. For instance, the functions at addresses 0x417b20, 0x41b2d0, 0x4190c0, 0x419ab0, 0x41a2b0, 0x41b4a0, 0x41be70 and 0x41c1f0 were referencing the main C++ file in the project. Fig. 8 shows one of these references.

Table 1. Identified Source Codes

Func. no.	Function ID @ Address	Source Code Reference	Match
70	[sub_406800] @ [0x406800]	complex.cpp	100%
146	[sub_409620] @ [0x409620]	lang.cpp	100%
159	[sub_40A1E0] @ [0x40a1e0]	matrix.cpp	100%
261	[sub_4119B0] @ [0x4119b0]	parser.cpp	100%
334	[sub_417B20] @ [0x417b20]	preccalc.cpp	100%

RESource has detected several math-related functions in the disassembly. The script has generated a comprehensive execution log that is self-explanatory and describes the analysis process. Even though the current version of RESource does not include heuristic query processing techniques, it is able to detect more than 70% of the original source files with an accuracy of 100%. Also, the script is useful for gaining insight into the functionality of the ASM file.

Concerning the Offline Analysis module, the current implementation includes a dictionary of common Windows APIs alongside with a brief description of each one. This dictionary was built with malware analysis in mind. Therefore, it includes about 200 of the most common kernel and user level functions known to be used by existing malware [13].

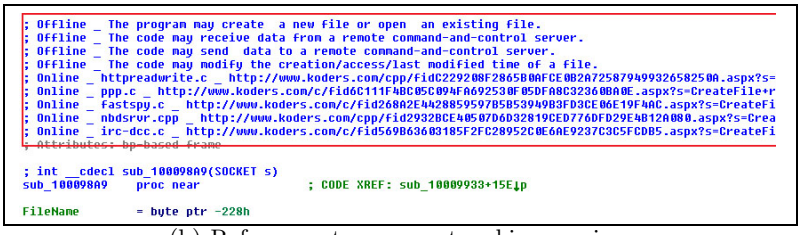
In our second scenario, we run RESource on several malware disassemblies. The Offline analyser helps the reverse engineers to understand network connectivity and data gathering functionalities of malware by adding relevant comments.

In Fig. 9 there are several routines of malware performing file I/O operations and communication with a remote command and control server. In cases where RESource returns results from both the online code repositories and the Offline analyzer, an emergent consistency is observed. As an example, Fig. 9(e) depicts a portion of assembly code that is capturing the screen and saves it to a file to be remotely transmitted. As can be seen, RESource gives reliable information

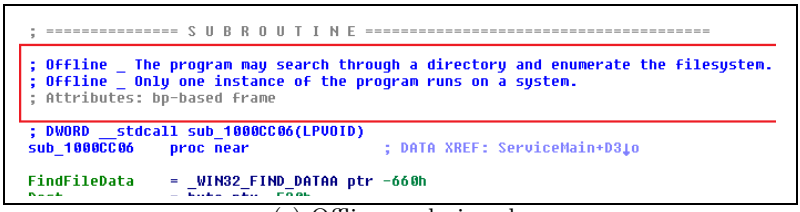
```
.text:0041B2D0 ; =============== S U B R O U T I N E =======================================
.text:0041B2D0
.text:0041B2D0 ; Online _ preccalc.cpp _ http://www.koders.com/cpp/fidC00F18FC54602A45811CF096F055D2
.text:0041B2D0 ; Attributes: bp-based frame
.text:0041B2D0
.text:0041B2D0 sub_41B2D0      proc near               ; CODE XREF: sub_4099D0+55↑p
.text:0041B2D0                                         ; WinMain(x,x,x,x)+312↓p
.text:0041B2D0                 push    ebp
.text:0041B2D1                 mov     ebp, esp
.text:0041B2D3                 call    sub_418EE0
.text:0041B2D8                 call    sub_4190C0
.text:0041B2DD                 call    sub_4188D0
.text:0041B2E2                 push    offset aPreciseCalcula
.text:0041B2E7                 push    1F8h
.text:0041B2EC                 call    sub_409440
.text:0041B2F1                 add     esp, 8
.text:0041B2F4                 mov     off_425710, eax
.text:0041B2F9                 mov     eax, off_425710
.text:0041B2FE                 push    eax
```

Fig. 8. Identified "precalc.cpp" file @ 0x41B2D0

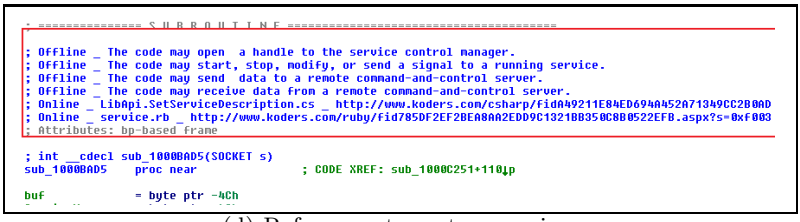
(a) Routine involving file I/O



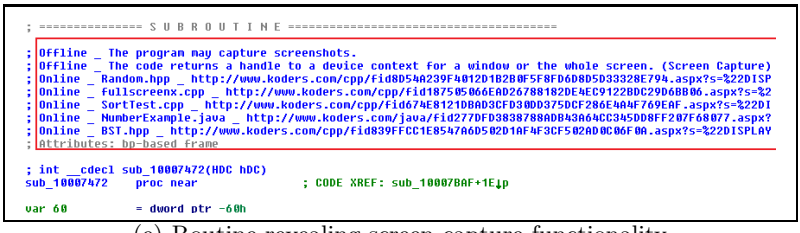
(b) References to some networking services



(c) Offline analysis only



(d) References to system services



(e) Routine revealing screen capture functionality

Fig. 9. Examples of the final outcome

in both Offline and Online comment sections. Such rich informal expression of a comment may really be beneficial for the hectic job of a reverse engineer.

Discussion
A side by side comparison between the outputs of RE-Google and RESource was not possible because the underlying search framework of RE-Google was deprecated. In other words, RE-Google is not functional anymore. RESource takes an intra-procedural approach to extract features and build queries. It could be argued that an inter-procedural approach could improve the accuracy of the Online analysis. However, the search engines provide limited commands for executing logic-based queries and some of them do not provide direct APIs to their repositories. Adopting a heuristic query building algorithm that tries different elements in the query string and selects the best match could improve the accuracy of the identified online projects. As to the accuracy of the Offline analysis, it clearly depends on the number and selection of the functions defined in the dictionary. In a situation where we have results from both the online and offline analyzers, the reverse engineer would have the maximum information. This happens when programs make use of standard libraries such as VCL or MFC. In other cases, there might be no results from the online module. This happens when malware authors use non-standard components or they use certain wrappers around standard system calls. Also, they might use non-standard low level kernel functions for performing simple I/O operations.

5 Conclusions

Software reverse engineering is a complex task. Applications like malware analysis can grasp the benefits of a tool able to automatically give reliable information about the matching between open source and assembly code.

In this paper, we established a framework to develop such a tool – RESource – that exploits some features existing at both the source and the assembly file levels. Based on these features, queries are triggered on certain online repositories used by the developers' community. If there is no query result, the tool is still able to provide some information about the functionality of a portion of the ASM file by a local offline analysis. The reverse engineer's task is thus greatly simplified.

References

1. Abrial, J.R.: The B Book - Assigning Programs to Meanings. Cambridge University Press (1996) ISBN 052149619-5
2. Boomerang: a general, open source, retargetable decompiler of machine code programs, `http://boomerang.sourceforge.net/`
3. Bryant, R.E., O'Hallaron, D.R.: Computer Systems – A programmer's Perspective, 2nd edn. Addison Wesley (2010) ISBN 0136108040

4. Eymery, D., Eymery, O., Borello, J.-M., Fraygefond, J.-M., Bion, P.: GenDbg: un débogueur générique. In: Symposium Sur la Séurité des Technologies de l'information et des Communications, SSTIC 2008, France (2008)
5. GDB: The GNU Project Debugger, http://www.gnu.org/software/gdb/documentation/
6. GNU Binutils, http://www.gnu.org/software/binutils/
7. Google Code, http://code.google.com/
8. Google Data APIs, http://code.google.com/p/gdata-objectivec-client/
9. IDA Pro multi-processor disassembler and debugger, http://www.hex-rays.com/products/ida/index.shtml
10. IDA Pro Re-Google Plugin, http://regoogle.carnivore.it/
11. Lagadec, P.: Dynamic Malware Analysis for Dummies. In: Symposium Sur la Sécurité des Technologies de l'information et des Communications, SSTIC 2008, France (2008)
12. Precise Calculator Project, http://sourceforge.net/projects/preccalc/
13. Sikorski, M., Honig, A.: Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press (2012) ISBN 1593272901
14. The Hex-Rays Decompiler, http://www.hex-ays.com/
15. Troshina, K., Chernov, A., Derevenets, Y.: C Decompilation: Is It Possible? In: Proceedings of International Workshop on Program Understanding, Altai Mountains, Russia, pp. 18–27 (2009)
16. Troshina, K., Derevenets, Y., Chernov, A.: Reconstruction of Composite Types for Decompilation. In: Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation, SCAM 2010, Timisoara, Romania, pp. 179–188 (2010)
17. Valgrind – a suite of tools for debugging and profiling, http://valgrind.org/
18. WinDbg debugger for Microsoft Windows, http://www.windbg.org/

Touchjacking Attacks on Web in Android, iOS, and Windows Phone*

Tongbo Luo, Xing Jin, Ajai Ananthanarayanan, and Wenliang Du

Syracuse University, Syracuse NY, USA

Abstract. To make it easy for applications to interact with the Web, most mobile platforms, including Android, iOS, and Windows Phone, provide a mechanism that allows applications to embed a small but powerful browser component inside. This mechanism is called WebView in Android (it is called different names in other platforms). WebView implements a number of APIs that can be used by applications to interact with the web contents inside WebView. It has been pointed out by the previous work that malicious applications can use these APIs to attack the web contents inside WebView. Proposals are made by the previous work to fix the problems of those APIs. We have discovered that by fixing those APIs, WebView is still not secure. This is because the previous work only focuses on the APIs specifically designed for WebView; they have overlooked the APIs that WebView inherits from its super classes. These APIs are designed for the general-purposed user interface (UI) components, and they seem to pose no risk to those components; however, the combination of these APIs with the Web has led to new risks. We have identified several attacks based on these APIs. Our attacks are called Touchjacking attacks. They treat WebView as a blackbox, i.e., they do not use the APIs that are designed specifically for WebView; instead, they only use the inherited APIs. Through these APIs, malicious applications can attack the web contents inside WebView. The impact of the attacks is quite significant, as all the platforms that we have studied, including Android, iOS, and Windows Phone, are vulnerable to these attacks.

1 Introduction

In most mobile platforms, including Android, iOS, and Windows Phone, web browser is not just a stand-alone application anymore, it can be incorporated into applications. This is achieved by exposing web browser as a reusable component that can be embedded by applications. Such a component is called `WebView` in Android, `UIWebView` in iOS, and `WebBrowser` in Windows Phone. For the sake of simplicity, we only use the term WebView throughout this paper.

WebView makes it very convenient for developers to integrate browser functionalities, such as web page rendering, navigation, and JavaScript execution,

* The project was supported by the Google Research Award and the NSF Award No. 1017771.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 227–243, 2013.

into their mobile applications. Applications requiring basic browser functionalities can simply include the WebView library and create an instance of WebView class. The use of WebView is pervasive. In the Android Market, 86 percent of the top 20 most downloaded Android apps in each of the 10 categories use WebView.

WebView APIs. There are two types of APIs in WebView. One type is the APIs implemented by the classes associated with WebView. These APIs are designed for applications to interact with the web contents. We call this type of APIs the web-based APIs. Examples of these APIs include `loadURL`, `addJavascriptinterface`, `CookieManager.getCookie`, etc. The attacks described in [7] target the web-based APIs. The other type of APIs are those inherited. WebView is a specialized user interface (UI) component, and like others, such as buttons and text fields, it is designed as a subclass of the more generic UI components, such as the `View` class. As a result, WebView inherits its super classes' APIs. We call this type of APIs the UI-based APIs.

To enable interactions, WebView implements several APIs, allowing mobile application code (from outside WebView) to interact with the web contents, and JavaScript code (from inside WebView) to interact with the mobile application contents. Luo et al. pointed out that these APIs, if not properly protected, can lead to security problems [7]. Luo et al. has studied how those malicious apps launch attacks on the web contents inside WebView by taking advantage of lacking of access control on those web-based APIs and hooks [7]. However, once a better access control is enforced on the communication channel, the attacks can be defeated which is not difficult to achieve. For example, the `loadURL` is one of the most dangerous APIs used in Luo's attacks, but it turns out that most applications do not use this API to inject JavaScript code into WebView. Therefore, an easy solution is to modify this API and restrict it to load URL only, instead of allowing it to inject JavaScript code.

Assume such an access control system can be implemented in WebView, and all the vulnerable APIs of WebView are protected, the question is whether WebView is safe now. A complete access control by WebView should control all the potential interaction channels between applications and WebView. No study has looked at whether the UI-based APIs inherited by WebView can pose risks to the contents that reside inside the webView. The objective of this work is to study the feasibility of attacks using the UI-based APIs. As Figure 1(a) illustrated, our attacks will not use any of the web-based APIs designed for WebView. In other words, even if the problems described in [7] are fixed, there are still ways to attack the contents in the blackbox WebView.

As we all know, when software components are reused (e.g., through libraries or class inheritance), their features, although safe and appealing for other systems, may bring danger to new systems. For WebView, it was not clear whether those inherited UI-based APIs pose any threat in the new systems, especially whether they can be used by malicious applications to attack the contents within WebView. There has been no study to investigate the security impact of those UI-based APIs inherited by WebView, mostly because these UI-based APIs have not appeared to

be problematic to other UI components. After studying WebView, we realized that the attacks conducted by Luo et al. only covered one type of interaction, not all.

Security Concerns on UI-Based APIs. From the security perspective, there is one thing that clearly separates WebView from the other UI components, such as buttons, text field, etc. In those UI components, the contents within the components are usually owned by or are intended for the applications themselves. For example, the content of a button is its label, which is usually set by applications; the content of a text field is usually user inputs, which are fed into applications. Therefore, there is no real incentive for applications to attack the contents of these components. WebView has changed the above picture.

In mobile systems, the developers of applications and the owners of web contents inside WebView are usually not the same. Contents in WebView come from web servers, which are usually owned by those that differ from those who developed the mobile applications. It should be noted that before Facebook released its own applications for iPhones and Android phones, most users used the applications developed by third parties (many are still using them). For example, one of the most popular Facebook apps for Android is called FriendCaster for Facebook, which is developed by Handmark, not Facebook. Because of such an ownership difference, it is essential for all mobile platforms to provide the assurance to web applications that their security will not be compromised if they are loaded into another party's mobile applications.

A WebView component with better access control enforced on all the cross-component communication channels can be treated as a blackbox. The mobile system guaranteed that the integrity and confidentiality of the web applications cannot be compromised even if they were loaded into the WebView embedded in a malicious application. Although users may not fully trust the third-party mobile apps, they fully trust the system once they make sure that they are using the WebView. The similar trust assumption is made when users view private contents in an iframe which is embedded in a third-party mashup web application. This is because users trust the isolation mechanism enforced by the browser to constraint the access from the host webpage if it comes from a different domain.

Overview of Our Work and Contribution. In this paper, we systematically studied the security impact of these UI-based APIs. Our attack model is the following: First, we assume that the mobile application is malicious; it embeds one or more WebView components in it. The target of the attack is the web contents within the WebView components. The attackers are interested in stealing sensitive information from the web page, or compromising the integrity of the web page and its interaction with user.

We further assume that the attackers can only use the UI-based APIs inherited by the WebView class. In other words, the malicious applications cannot directly interact with the web contents inside WebView. This assumption will significantly distinguish our work from that by Luo et al. [7], which focuses only on the web-based APIs. Putting this assumption in a different way, we are

investigating whether WebView can be secured if it is redesigned to address the attacks in [7].

We have identified several different attacks that can be launched on WebView solely using the UI-based APIs. We have studied these attacks in three popular mobile phone platforms, including Android, iOS, and Windows Phone. All of them are vulnerable to our attacks. For the sake of simplicity, we will only talk about WebView and the attacks in the context of Android.

Our discoveries are significant, as they demonstrate that securing and redesigning the web-based APIs are not sufficient; we also need to study the APIs that WebView inherited from its super classes, understand how dangerous they are to the web contents inside the WebView, and find solutions to secure them. This paper only focuses on the attack part; developing solutions to solve the problem for Android, iOS, and Windows Phone is still a work-in-progress, and will be published in our future papers.

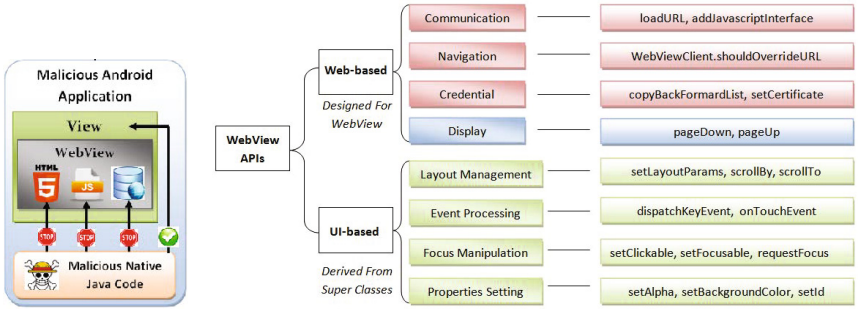
2 WebView APIs

To enable applications to browse the Web from within themselves, instead of using an external browser application, Android provides a package called `android.webkit` to applications. This package contains several classes, each for different purposes. The most important class among them is called `WebView`, which is a View class that displays web pages. This class is the basis for displaying web contents within applications. It uses the WebKit rendering engine to display web pages; it also includes methods to navigate forward and backward, zoom in and out, perform text searches, etc. In addition to WebView, `android.webkit` also includes several other classes related to the Web, such as `CookieManager` (for managing cookies), `WebViewClient` (for customized handling of events within WebView), `WebViewDatabase` (for managing the WebView database), etc.

Jointly, these classes expose many APIs to Android applications. Based on their purposes, these APIs can be divided into two main categories (see Figure 1(b)). One category is the APIs that are designed for the control of web pages and their related data (e.g. cookies, histories, and caches), and we call them the web-based APIs. The other category is the APIs that are derived from their super classes, which are designed for the general user interface (UI) components, and we call them the UI-based APIs.

2.1 Web-Based APIs

The classes in the `android.webkit` package jointly expose a number of APIs to the applications for better manipulation and control over the web contents inside WebView. Those APIs are quite useful for application developers to embed and customize "browser-like" components within applications, and thus enrich the functionalities of applications. We will not go over all those APIs; we only describe those that are related to security.



(a) Threat Model (b) WebView APIs Overview

Fig. 1. WebView APIs

Webpage-Android Communication. Android WebView provides a bidirectional communication channel between the webpage environment inside WebView and the native Android application runtime. For example, WebView provides a mechanism for the JavaScript code inside it to invoke Android apps' Java code. The API used for this purpose is called `addJavascriptInterface`. In addition to the JavaScript-to-Java interaction, WebView also supports the interaction in the opposite direction, from Java to JavaScript. This is achieved via another WebView API `loadUrl`.

Webpage-Related Hooks. Android applications can monitor the webpage navigation and rendering events occurred inside WebView. This is done through the hooks provided by the `WebViewClient` class. These hooks will be triggered when their intended events occur inside WebView. Once triggered, these hooks can access the event information, and may change the consequence of the events. For example, by overloading the hook `shouldOverrideURL`, Android applications can intercept and modify the destination URL when the user tries to navigate to another web page or site.

Webpage Credentials. All the credentials and private data of web pages are stored in an internal database, which is isolated from Android applications. However, WebView exposes many APIs to allow applications to fetch or modify the sensitive webpage contents in the internal database. For example, Android applications can directly inject the certificate of a webpage through the API `setCertificate`, cookies can be accessed using `CookieManager.setCookie`, and so on.

2.2 UI-Based APIs

The `android.webkit` package includes a number of classes, most of which inherit directly from `java.lang.Object`, which is the root of all classes in Java. The APIs inherited from this root class do not pose much risk. An outlier among these classes is the `WebView` class, which is the main UI class in the

package. This class inherits the APIs from several classes. Its inheritance tree is the following (starting from the root): `java.lang.Object`, `android.view.View`, `android.view.ViewGroup`, and `android.widget.AbsoluteLayout`. Moreover, `WebView` also implements seven interfaces, with six of them coming from the `android.view` package, and one from `android.graphics` [1].

Among all the classes and interfaces inherited by `WebView`, the most significant class is `Android.view.View`, which is commonly used by Android applications. The View class represents the basic building block for user interface components; it usually occupies a rectangular area on the screen and is responsible for drawing and event handling. This class serves as the base for subclasses called `widgets`, which offers fully implemented UI objects, like text fields and buttons. `WebView` is just a customized widget.

Our attacks focus on the APIs provided by `Android.view.View`. These APIs can be classified into several categories, including Layout Management, Event Processing, Focus Manipulation, and Properties Setting, all of which are the basic functionalities designed for native Android UI objects. We will illustrate some of the commonly used APIs in this `View` class. It should be noted that some of the APIs inherited from the `View` class are overridden in the `WebView` class, but we still count them as the UI-based APIs.

Layout Management. One of the basic features of Android UI objects is to provide basic methods to handle the screen layout management. For example, a view object has a location (expressed as a pair of left and top coordinates) and two dimensions (expressed as a width and a height). Android applications can use the methods, such as `layout`, `setX`, and `setMinimumHeight`, to configure locations. It is possible to retrieve the location of a view object by invoking the methods `getLeft` and `getTop`. Similar methods can also be used to get the size information of the WebView.

Android also provides basic supports for views that need scrollbars. This includes keeping track of the X and Y scroll offsets as well as drawing scrollbars. Using the methods like `scrollBy` and `scrollTo`, Android applications can control the displayed area of the content in the view object. Obviously, for WebView, the contents inside the WebView are web pages.

Event Processing. Each Android view object is responsible for drawing the rectangular area on the screen that it occupies, and handling the events in the area. Views allow clients to set listeners through hooks that will be notified when something interesting happens to the view. For example, by using the method `setOnKeyListener`, Android applications can register an event handler callback function which will be invoked when a key is pressed in this view. Besides intercepting the events, the view class also exposes methods for Android applications to pass motion events down to the target view.

Focus Manipulating. The Android framework will handle moving focus in response to user input. To force focusing on a specific view, applications can call `requestFocus()` of that view.

Properties Setting. Other advanced features related to appearance could be the background color or alpha property of WebView, like methods `setBackgroundColor` and `setAlpha`.

2.3 Attack Model

For all of the attacks described in this paper, we have the following assumptions:

1. **We are concerned about potential malicious applications in mobile devices.** As we pointed out, the developers of the apps and the owner of the web contents inside WebView are usually not the same. Our investigation shows that among the top 113 apps that use WebView, 49 are third-party apps. Therefore, it is quite common for web contents to be loaded into an untrusted environment.
2. **We assume that the users clearly know they are using WebView.** Users make sure they are using the secured blackbox WebView instance to access web contents, and they trust that the mobile system can isolate the contents inside WebView from those from outside.
3. **We assume that the effective access control mechanism is already enforced on the Web-based APIs exposed by the WebView.** As we mentioned before, Web-based APIs are powerful to control the web contents inside WebView. We assume a perfect redesigned access control model has been implemented on WebView to isolate the contents inside WebView from outside world. This assumption clearly distinguish this work from that in [7], because under such an assumption, the attacks describe in [7] will not be threats any more.
4. **We assume that the UI-based APIs are accessible by the apps.** WebView is a specialized user interface (UI) component, and like others, such as buttons and text fields, it is designed as a subclass of the more generic UI components, such as the `View` class.
5. **We assume that malicious apps are only granted with one permission.** It should be noted that to successfully launch the attacks described in this paper, malicious Android applications only need one permission `Android.permission.INTERNET`. This permission is widely granted to 86.6% of free (and 65% of paid) Android applications [4]. Generally speaking, these attacks are relatively easy to launch and difficult to detect, since they only require one very common and less-dangerous permission.

3 Touchjacking Attacks

In this section, we describe how to let users generate touch events, and how to hijack those events for malicious purposes. We call this type of attacks the `Touchjacking attack`. We describe three attacks; based on their different attack strategies, we give them different names.

We give a brief overview of the three attacks here, and explain the details later in this section. Figure 2 illustrates the attacks.

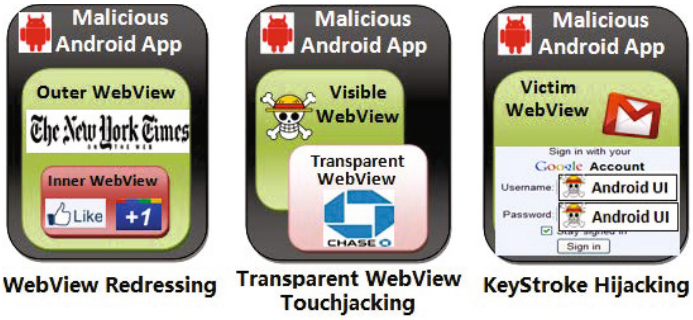


Fig. 2. Touchjacking Attack Overview

1. **WebView Redressing Attack.** In this attack, malicious applications put a smaller WebView on top of a larger one, making the smaller one look like an element (e.g. button) within the larger one.
2. **Invisible WebView Attack.** In this attack, malicious applications overlay an invisible WebView on top of a visible one, causing users to see the visible one, but operate on the invisible one.
3. **Keystrokejacking Attack.** In this attack, malicious applications overlay some native UI objects on the top of the HTML elements inside WebView; while the user believe that they are typing in the field that belongs to a web page, they are actually typing in a field that belongs to the malicious applications, which can steal the information typed by the users.

3.1 Positioning Method

By default, after being loaded into a WebView, the webpage will be displayed inside the WebView. If the size of the webpage is larger than the size of the WebView, only the most top-left area of the webpage will be displayed initially. However, in order to carry out our attacks, certain HTML elements (e.g. a button) of the targeted webpage must be carefully positioned. Only using the traditional positioning methods that facilitate clickjacking attack in browsers is not enough to meet the positioning requirement for Touchjacking attacks. We describe some positioning techniques.

Pixel Coordination. Android applications can use the following APIs to position a web page to a specific position inside WebView: `scrollBy`, `scrollTo`, `pageDown`, `pageUp`. The method `scrollBy(x,y)` scrolls the page by x pixels horizontally and y pixels vertically; the method `scrollTo(x,y)` scrolls the page to the `(x, y)` position. The method `pageDown` and `pageUp` scroll the display area to the top and bottom of a webpage. Attackers can also use the websetting APIs to change the font size or zoom level of the webpage, such as `setTextSize` and `setDefaultZoom`.

URL Fragment Identifier. Using pixel coordinates to position a target can be inaccurate due to other factors, such as rendering differences between browsers

and font size differences between platforms. A solution to this problem is to use the URL fragment identifiers to position anchor elements of the webpage. Anchors and URL fragments are commonly used together to link to a particular section of the text within an HTML document. When a URL containing a fragment identifier is loaded, a browser will scroll the page so that the anchor is at the top of the viewable area. An anchor can be created in two ways, either by adding a `'name'` attribute to an `'a'` tag, or by adding an `'id'` attribute to any element. The following example shows how to navigate to the specific `div` tag using URL fragment identifiers.

3.2 WebView Redressing Attack

Generally speaking, the idea behind the WebView redressing attack is to seamlessly merge two or more WebView containers, making them look like one. When the non-suspicious user reacts to the contents inside WebView by clicking some links or buttons, because what the user clicks on may belong to a different page in another WebView, the user is tricked into reacting to the contents in another WebView, and those contents are not even displayed to the user.

The attack consists of two or more WebViews (we will use two in our description). One of the WebViews is called the outer WebView, and the other is called the inner WebView. The inner WebView loads the malicious webpage, and it is intentionally made small, so it only displays a very small portion of the webpage to users. This is important, as the attackers do not want the users to see the entire page, which reveals the malicious intents. The malicious application can use the positioning method described above to display a specific part of the page (such as a button) to users.

The outer WebView is larger, and is for the users to view web contents. Attackers overlay the inner WebView on top of the outer WebView, and make it cover a selected area of the outer WebView. Because the inner WebView is small and has no obvious boundaries, the inner WebView looks like part of the elements on the webpage inside the outer WebView. If users react to the contents in the outer WebView, and clicks on the buttons within the inner WebView, they are actually reacting to the contents in the inner WebView. This is dangerous, as the users never got a chance to see the contents that they have reacted upon.

Case Study. We demonstrate the WebView redressing attack using an example. Facebook has been a major spam target; one of the goals of the spammers is to find ways to post links or other information on Facebook user's walls. Just like email spams, no matter what improvement the company makes, spammers have always been able to find new ways to cause problems. We will demonstrate a new way to launch the "likejacking attack" [14] by using the WebView redressing technique, so that the users can be tricked into "Like"ing spam pages.

In this attack scenario, assume that the malicious Android application is written for New York Times. Normally, only the outer WebView is visible and users will use this WebView to visit the articles at www.nytimes.com (see Figure 3(a)). The malicious Android application can insert the inner WebView at any time

when the user navigates to the New York Times page. The inner WebView contains the spam article, with a Like button (see Figure 3(b); we did not show the spam article in the figure). The attackers need to pre-calculate the location of the inner WebView (Figure 3(b)) to redress the webpage inside the outer WebView.

After the redressing, what the user sees is shown in Figure 3(c). Clearly, it is quite difficult for the user to see that the Like button is not part of the New York Times page. If the user really likes this article and wants to share it through Facebook, he/she will click on the Like button, not knowing that the button is associated with a different article hidden in another WebView.

If the user has not logged into Facebook yet from this application, once clicking on the Like button of the inner WebView, a dialogue window (which is a new WebView instance) will be popped up with the Facebook's login page inside (see Figure 3(d)). Since it is hard for the user to realize that the dialog window is not popped up by the outer WebView, the user may very likely log into Facebook, and eventually share the article that he/she has never seen.

It is also likely that the user may have already logged into Facebook from the inner WebView (due to the clicking of some legitimate Like buttons). Because cookies are shared among all the WebView instances within the same Android application, clicking on the Like buttons in another WebView will not result in the pop-up dialogue window; instead, the "like" request will be automatically sent to Facebook with the valid cookies.

(a) Outer WebView (b) Inner WebView (c) What User Sees (d) Login Dialog

Fig. 3. WebView Redressing Attack Example

3.3 Invisible WebView Attack

Both Android and iOS systems allow applications to set transparency on WebView (UIWebView) objects. Low opacity may result in the webpage inside WebView being hardly visible, or completely invisible. In Android 3.0, applications can use the method `setAlpha` to set the opaque level of the WebView object. Every native Android UI object maintains the alpha property and exposes the setter and getter to applications. Since the `WebView` class was derived from the `View` class, it also inherits this property. It should be noted, when a WebView object is transparent (i.e. alpha value equals to 0), it is transparent visually, not physically, i.e., users can still touch/click on the page inside a transparent WebView. The following code demonstrates how to set the WebView transparent.

```
WebView mWebView = (WebView) findViewById(R.id.webview);
mWebView.setAlpha(0);
```

The transparency feature is intended for generic UI components, and it brings no harm to them; however, when this feature is inherited by WebView, it poses great danger to the web contents inside WebView. We describe how this feature can be used for attacks.

In this attack, malicious Android applications need to have two WebView instances: one visible and the other invisible. The `visible` WebView will load an attractive webpage that is controlled by attackers, and the purpose of this page is to entice users to perform touch actions. For example, this web page can be a small game. Another WebView is `invisible`, and it loads the targeted webpage. The invisible WebView is put on top of the visible one. Therefore, when the user touch something that is apparently in the visible WebView, the touch actually goes to the invisible one, because it is on the top.

To successfully launch the touchjacking attack, attackers need to first calculate the position where user may perform the touch action. Since the attacker controls the visible webpage, it is not hard to predict the position and precisely overlay the UI in the targeted webpage inside the invisible WebView object on top of specific position. Attackers can use the positioning techniques mentioned in the beginning of this section to control the place of the clickable elements (e.g. button, link).

Case Study 1. In this attack example, we repeat the case study in the previous subsection, but using the transparency technique to achieve the same goal. We assume that the malicious Android application is written for New York Times, and the user is currently reading an article from there. This time, the article itself has a legitimate Like button to facilitate sharing via Facebook (see Figure 4(a)).

Attackers create another WebView (invisible), and load the spam page inside it. This page contains an article that the spammers want the user to share with their Facebook friends, and there is a Like button on this page (Figure 4(b) shows this spam page, but we did not show the spam article inside).

The malicious application then overlays the invisible WebView on top of the visible one. Using the positioning techniques, the attackers can make the two

(a) Visible WebView (b) Invisible WebView (c) User's View

Fig. 4. Invisible WebView Attack Example

Like buttons in both WebViews be placed at exactly the same location on the screen, i.e., they completely overlap. Because of the transparency, what the user sees is exactly the same as that in Figure 4(a).

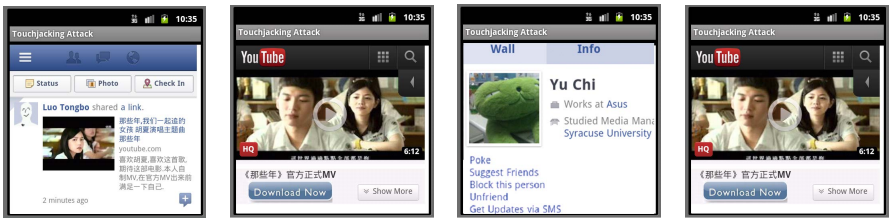
When the user clicks on the Like button, the click event goes to whatever is on the top, i.e., the transparent WebView, not the one for New York Times. As results, the spam article is shared to the user's Facebook friends. This consequence is the same as that in the WebView redressing attack.

Case Study 2. If the user also uses the malicious application to log into his/her online accounts (such as Facebook), the attack can be much more severe. We use Facebook to demonstrate how to use the Invisible WebView attack to hijack the touch events and trick users into deleting friends from their Facebook accounts.

Before the attack is launched, users have logged into their Facebook accounts from the visible WebView, and are viewing their Facebook pages (Figure 5(a)). At this time, the invisible WebView is not overlaid yet. When the user clicks a link shared by his/her friend, WebView will navigate to another webpage (Figure 5(b)); this webpage is not malicious, but the attacker needs to know the possible click points. At the same time, the application needs to overlay the invisible WebView on the top, and inside the WebView should be the Facebook webpage (Figure 5(c)).

Attacker can also precisely put the `UNFRIEND` link of the transparent Facebook page on the top of the `DOWNLOAD` button of the visible WebView. If the user wants to download the video as shown in Figure 5(d), the user needs to click the `DOWNLOAD` button. Because the `UNFRIEND` button is on the top, this button is actually clicked, and user's some friends will be deleted from the friend list.

Although the user has never actually logged into Facebook account using the invisible WebView, since the cookies are shared among all WebView instances within the same Android application, the UNFRIEND request from the webpage in the invisible WebView will be able to attach the Facebook cookies and cause the deletion of the user's friend.

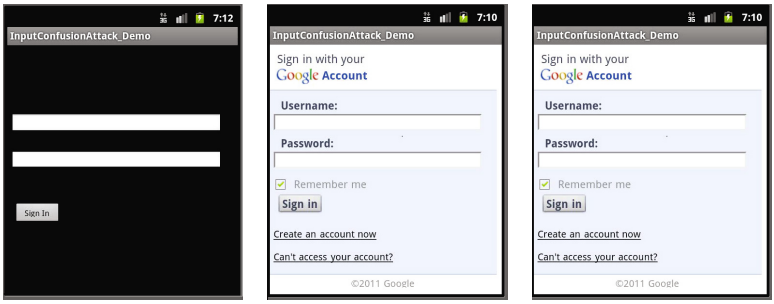


(a) Initial Visible WebView (b) Visible WebView (c) Invisible WebView (d) What User Sees

Fig. 5. Invisible WebView Attack Example

3.4 Keystroke Hijacking Attack

In the previous attacks, attackers redirect the user's actions toward the webpage in a WebView instance that is different from what the user sees. In this attack,



(a) Native UI Only (b) WebView Only (c) User's View

Fig. 6. Keystroke Hijacking Attack Example

we will demonstrate how attackers can redirect those actions to the native Android UI objects (e.g. a text field) that is completely controlled by the malicious applications. If the user's actions involve secrets (e.g. passwords), the attacker can get the secrets.

The attack is based on the fact that the HTML UI objects inside WebView and the Android native UI objects are based on the same GDI (`skia`), and the exterior appearance of the HTML UI objects look similar to their related native UI objects. For example, the HTML input field looks almost the same as the text editing widget `EditText`, which is a native UI component of Android. Therefore, if we put a native UI object on top of the HTML UI object of the same type, users will not be able to tell the difference. If they decide to type into what appears to be a part of the webpage, they will be typing into the native UI object that belongs to the attackers.

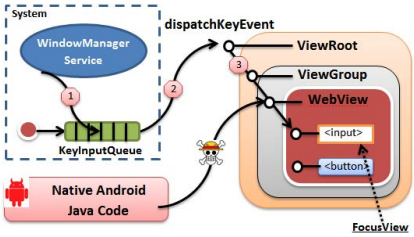
To successfully launch the attack, the attackers should precisely overlay the native Android UI objects on top of the HTML objects of the web page inside WebView, with exactly the same size and location. Since the layout of the victim page is almost stable in many cases (e.g. login pages), attackers can quite easily calculate the size and position of the targeted UI objects within the webpage.

Case Study. We use Gmail as an example to demonstrate how the attack works. We separately display the two layers of layout in the malicious applications. Figure 6(a) is the upper layer, consisting of two `EditText` native UI components. Figure 6(b) is the lower layer, consisting a WebView with the Gmail login page inside. When being displayed on the screen, the two `EditText` UI components will exactly overlap with the two input fields on the Gmail login page. When users type the username and password, they actually type in the `EditText` UI components, which are accessible by the attacker.

Users may be aware of the attack once they finish the input actions and submit the form, because the actual HTML input objects are empty, and an error message will be displayed. To further disguise that, the attackers should also add a fake submit button (native UI object). Once the fake button is clicked, the malicious application should ask WebView to navigate to an error page, displaying

API Name	Event Type	Action
dispatchDragEvent	DragEvent	Drag
dispatchKeyShortcutEvent	KeyEvent	Key
dispatchKeyEventPreIme	KeyEvent	Key
dispatchHoverEvent	MotionEvent	Touch
dispatchTrackballEvent	MotionEvent	Trackball
dispatchKeyEvent	KeyEvent	Key
dispatchTouchEvent	MotionEvent	Touch

(a) Event Dispatching APIs



(b) Event Dispatching Mechanism

Fig. 7. Event Dispatching Mechanism and APIs

something like "Page cannot be displayed due to network problems". After the users go back to the previous page, the malicious application remove all the overlaid native UI objects, so the users can proceed without raising suspicions.

4 Event-Simulating Attacks

The touchjacking attacks described earlier hijack real user's touch events, while this attack can generate fake touch events. As we have discussed in Section 2, like all of the view-based Android UI objects, the WebView class inherits a number of methods from the View class, including the ones needed by the event-dispatching mechanism in Android. Those APIs are listed in Figure 7(a). However, those event-dispatching APIs also have to be exposed to Android applications. As results, by invoking those APIs to dispatch the action to the currently focused HTML UI objects, applications can generate keystroke, click, and touch-screen events within WebView without consents from users (Figure 7(b)).

Since we believe that those APIs directly interact with the web page inside the WebView, attacks using those APIs will not treat the WebView as the blackbox. Although this attack is more powerful than the touchjacking attacks, we believe in the future, those APIs will be blocked by the android system. Due to the page limitation, we cannot cover the details of this attack here.

5 Attacks on Other Platforms

To see whether the attacks we identified in this paper work on the platforms other than Android, we have tried the attacks on iOS (version 4.3.2) and Windows Phone 7. All the three types of Touchjacking attacks work on iOS and Windows Phone 7. For the event-simulating attack, unlike Android, iOS does not provide APIs to dispatch key/touch events to UIWebView. Therefore, we were not able to directly simulate key/touch events in UIWebView. Similar to iOS, Windows Phone does not provide any API support for programmatically invoking an event.

6 Related Work

Since the first bug report on the negative usage of iframe by [11], Clickjacking attack [6], UI redressing attack [9], and Next Generation Clickjacking attack [15] have been developed by taking advantage of transparent iframes. To successfully launch a clickjacking attack, a malicious page is constructed by attackers in a way that tricks victims into clicking on the elements in a different page that is only barely visible or completely invisible [16]. All of these attacks can be analysed as confused deputy attacks, where the user acts as the confused deputy [2].

Clickjacking can also be successfully launched on the browsers for mobile platform. Moreover, more confused deputy attacks targeting on mobile devices were proposed. The project [13] introduced the `Tapjacking` attack, which display a fake view (called a Toast) by generating a customized notification to pass interaction events such as finger taps to a hidden user interface overlaid underneath. To allow developers to prevent TapJacking, Android 2.3 provides a feature for Views to be only interactive when it is visible by explicitly setting the filterTouchesWhenObscured property to true.

Similar to Clickjacking and Tapjacking, the Touchjacking described in our paper can be counted as confused deputy attack as well. But our work is the first to study the attacks using WebViews, instead of Iframes or Toast. Although the goal of Touchjacking is to cause confused deputy and trick users misusing authority, the vulnerabilities that Touchjacking takes advantage of are completely different from other attacks. In other words, Touchjacking exploits the design flaw of WebView component. Therefore, our attacks cannot be prevented by the solutions proposed for clickjacking attacks, such as frame busting [12] and X-Option-Header [5]. Since this paper only focuses on the attack, details of the solutions are not included; they can be found in the work called Medium framework [8], which introduces the concept `Visual Integrity`. The XSS attack compromises the data integrity of particular web site, but Touchjacking and various kinds of Clickjacking attacks only compromise the visual integrity of the page loaded in the web container.

Touchjacking attacks differ from phishing [10] attacks because they do not trick users to enter secret credentials into a spoofed website. Instead, users need to enter their credentials into the real website in the WebView to establish an authenticated session. The attack can proceed until the user's session expires. The user's sensitive information is completely isolated from the malicious Android application throughout the attacks.

The project [3] discovered several phishing attacks that can be mounted against control transfer. Those attacks will take advantage of the lack of secure application identity indicators in mobile operating systems and browsers, so the user cannot always identify whether a link has taken him/her to the expected application. Some of the attacks target the phishing attack on browsers and WebViews, but they depend on faking the whole browser or using the WebView APIs to directly compromise the webpage. The spoofing attack we introduced in this paper is not due to any of the four control transfer scenarios, but a new control transfer from webpage to system.

Our paper is distinguished from the work [7], which tries to exploit WebView vulnerabilities by directly manipulating the contents inside WebView through the powerful APIs and hooks exposed by WebView. In this paper, we assume that all the access paths to directly communicate with the webpages inside WebView have been blocked or securely controlled.

7 Conclusion

The security problem of the WebView technology has been studied before, but the existing work focuses on how the APIs designed specifically for WebView can be abused to compromise the security of the web contents inside WebView. The work calls for adding extra access control into those APIs. This paper points out that even if those APIs are secured, WebView is still dangerous. This is because WebView inherits many UI-based APIs from its super classes, and those APIs can be abused as well, although in a very different way. We describe several attacks based on these APIs. We show that using these APIs, attackers can compromise the integrity and confidentiality of the web contents inside WebView.

References

1. Android-Team. Webview class reference, `http://developer.android.com/reference/android/webkit/WebView.html`
2. Close, T.: The confused deputy rides again! (2008), `http://waterken.sourceforge.net/clickjacking/`
3. Felt, A., Wagner, D.: Phishing on mobile devices. In: Web 2.0 Security and Privacy (2011)
4. Felt, A.P., Greenwood, K., Wagner, D.: The effectiveness of application permissions. In: Proceedings of the 2nd USENIX Conference on Web Application Development, WebApps 2011, Berkeley, CA, USA, p. 7 (2011)
5. Firefox. The x-frame-options response header, `https://developer.mozilla.org/en/The_X-FRAME-OPTIONS_response_header`
6. Hansen, R.: Clickjacking, `http://ha.ckers.org/blog/20080915/clickjacking/`
7. Luo, T., Hao, H., Du, W., Wang, Y., Yin, H.: Attacks on webview in the android system. In: Proceedings of the 27th Annual Computer Security Applications Conference, pp. 343–352. ACM (2011)
8. Luo, T., Jin, X., Du, W.: Mediums: Visual integrity preserving framework. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy, CODASPY 2013 (2013)
9. Niemietz, M.: Ui redressing: Attacks and countermeasures revisited. In: CONFidence 2011 (2011)
10. Niu, Y., Hsu, F., Chen, H.: iphish: phishing vulnerabilities on consumer electronics. In: Proceedings of the 1st Conference on Usability, Psychology, and Security, pp. 10:1–10:8. USENIX Association, Berkeley (2008)
11. Ruderman, J.: Bug 154957 - iframe content background defaults to transparent (2002), `https://bugzilla.mozilla.org/show_bug.cgi?id=154957`
12. Rydstedt, G., Bursztein, E., Boneh, D., Jackson, C.: Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In: IEEE Oakland Web 2.0 Security and Privacy (2010)

13. Rydstedt, G., Gourdin, B., Bursztein, E., Boneh, D.: Framing attacks on smart phones and dumb routers: tap-jacking and geo-localization attacks. In: Proceedings of the 4th USENIX Conference on Offensive Technologies, pp. 1–8. USENIX Association (2010)
14. Sophos. Facebook worm - likejacking (2010), http://nakedsecurity.sophos.com/2010/05/31/facebook-likejacking-worm/
15. Stone, P.: Next generation clickjacking (2010)
16. Zalewski, M.: Browser security handbook (2008), http://code.google.com/p/browsersec/wiki/Part2

Short-Term Linkable Group Signatures with Categorized Batch Verification

Lukas Malina[1], Jordi Castellà-Roca[2], Arnau Vives-Guasch[2], and Jan Hajny[1]

[1] Department of Telecommunications, Brno University of Technology,
Purkynova 118, Brno, Czech Republic
{malina,hajny}@feec.vutbr.cz
http://crypto.utko.feec.vutbr.cz
[2] Department of Computer Engineering and Mathematics,
Universitat Rovira i Virgili,
Av. Paisos Catalans 26, Tarragona, Catalonia, Spain
{jordi.castella,arnau.vives}@urv.cat

Abstract. In ad hoc wireless networks like Vehicular ad hoc Network (VANETs) or Wireless Sensor Networks (WSN), data confidentiality is usually a minor requirement contrary to data authenticity and integrity. Messages broadcasted from a node to other nodes should be authentic but also keep user's privacy in plenty scenarios working with personal data. Group signatures (GS) are used to provide privacy and authenticity to the users. Moreover, GS with batch verification can be efficient. Nevertheless, the current solutions have practical drawbacks like using an expensive tamper-proof hardware, the computation bottlenecks of the verification and revocation phases, complicated certificate distribution/revocation or omitting important properties like short-term linkability which is demanded in several applications, e.g. change lanes of vehicles in VANETs. To our best knowledge, our solution employs the short group signature with short-term linkability and categorized batch verification for the first time. Our solution provides more efficient signing and verification than compared schemes. Moreover, the solution allows secure and practical registration and revocation of users. The usage of proposed scheme protects the honest users who can now join and securely communicate without losing their privacy.

Keywords: Security, Group Signature, Batch Verification, Privacy, Efficiency, Ad hoc Wireless Network, Short-term Linkability.

1 Introduction

There are a lot of practical and theoretical scenarios where data confidentiality is not too important like data authenticity, integrity and user privacy during communication. For example, privacy is demanded by users in Wireless Body Sensor Networks (WBSN) where nodes are bound to human in order to measure medical data and user position [1]. In WBSN, the users concern about their potential monitoring by a malicious observer. Further, the received messages

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 244–260, 2013.

carrying useful data from several tens of nodes must be verified as soon as possible. The same problem with privacy arises in VANETs. For better intuition, we apply our proposed security solution to VANETs. Nevertheless, the solution can be applied to systems where the users' privacy, data authenticity and integrity are required during dense communication.

The wireless communication among vehicles brings many applications which can help drivers, prevent accidents or reduce traffic. A vehicular ad hoc network measures useful data like speed, location, road condition or alerts and distributes them using an On Board Unit (OBU) in a vehicle in order to increase security on roads and reduce traffic jams. OBU can be an embedded device/a user smartphone or a navigation with VANET application. Self-organized VANET offers two types of communication: wireless communication between a vehicle and a vehicle (V2V), and communication between vehicles and the VANET infrastructure (V2I) represented by Road-Side Units (RSU) which are connected to a fixed infrastructure (eg. Internet). Security in VANETs plays a key role in protecting against bogus and malicious messages, misusing at roads, eavesdropping etc. Common solutions guarantee the message integrity, authentication and non-repudiation. Furthermore, privacy is required due to the possibility of drivers tracking by malicious observers. Moreover, VANETs can serve in a dense urban traffic where hundreds of vehicles communicate in V2V or V2I, so that the security overhead and computation time must be minimal. In this case, the following scenario is considered: *Scenario 1*: A driver, Alice (A), with the car no. 2, which is depicted in Fig.1, can register special events (accidents, traffic jams, roads under construction etc.). Depending on the type of event, A immediately broadcasts a warning message through the wireless V2V communication to all participated cars in VANET. In this scenario, an accident is depicted in Fig.1. Supposing another driver, Bob (B), with the car no. n-1, who is in range and coming closer to A, receives this message. B also receives more messages from another cars in the area. Moreover, other messages can contain contradictory warnings or can be bogus. In short time, B must consider the validity of these

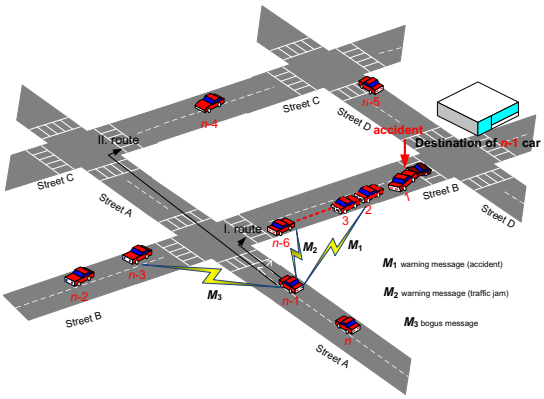


Fig. 1. The VANETs in urban traffic - Scenario 1

messages and quickly decides changing the route (from planed I. to II.). If B makes the right decision, he can avoid the situation referenced by the first warning message. It is obvious that the decision must come in real time and as soon as possible. In another case, a lot of cars periodically broadcast informations about them (speed, direction, location, break alerts etc.) and road conditions (changing of road lanes, distance between cars etc.) each other. The type of information depends on the application used by the car, but message processing must be efficient because the sending period of beacon messages is less than 300 ms [2].

The security proposals are challenged to connect privacy, security, efficiency and capable management in huge vehicular networks. The open problem of *Scenario 1* is how a lot of anonymous messages can be verified in real time. The related work tries to solve this problem using the batch verification of group signatures. But this approach takes more time than expected if the number of malicious messages appeared in batch is $\geq 15\%$ from all messages as is claimed in [3]. In order to improve this issue, we propose a novel solution with categorized batch verification with short-term linkability which can serve to recognize the malicious messages and excludes them from batch. Moreover, the short-term linkability significantly improves the signing phase, so that our scheme provides more efficient signing and verification than related works using GS.

The paper is organized as follow: The next section presents the related work which is focused on the security and privacy protection in VANETs and our contribution is outlined. Then, section 3 presents a basic scheme description, requirements and main cryptographic techniques used in our proposal. Further, section 4 introduces our solution and the phases of our scheme are described. The important phases like signing and verification are evaluated and compared with related solutions in section 5. Section 5 also contains the security consideration. Finally, conclusion and future work are presented in section 6.

2 Related Work and Our Contribution

In this section, we outline the related work and our contribution.

2.1 Related Work

Generally, the protection of privacy in VANETs can be ensured by three approaches, i.e., pseudonyms, group signatures and hybrid schemes. Anonymization through pseudonyms has been proposed in [4] and [5]. The work [6] uses anonymous certificates which are stored in vehicles (usually in a tamper-proof device). This approach uses a set of short-lived pseudonyms and privacy among vehicles is provided by changing these certified public keys. Nevertheless, in large urban VANETs, this approach is burdened by preloading and storing a large number of anonymous certificates with pseudonyms.

Group signatures (GS) in VANETs provide user anonymity by signing a message on behalf of a group. GS guarantee the unlinkability of honest users

and the traceability of misbehaving users. The scheme [7] called GSIS uses the combination of a group signature based on [8] with a hybrid membership revocation mechanism in the V2V communication, and Identity Based Group Signature (IBGS) in the V2I communication. The hybrid membership revocation with the list of revoked members (RL) works with a threshold value T_τ. In case $|\text{RL}| < T_\tau$, the scheme uses revocation verification algorithm, otherwise, the scheme updates the public/private group keys of all non-revoked members. For efficient verification, the authors of [9] propose a GS with batch verification in V2I which takes three pairing operations. This scheme called IBV has several drawbacks such as using tamper proof devices, it is vulnerable to tracking or impersonation attacks, see [10] for a complete description. The works [11] and [12] can efficiently verify a large number of messages in V2V. These schemes use short group signatures with fast batch verification (only two pairing operations are used instead of 5 n, where n is number of messages). Nevertheless, the performance of batch verification degrades in dense V2V communication with bogus messages. The On Board Units (OBUs) must process the messages quickly, they have between 100 ms and 300 ms to process a message [2]. Thus, the computation of expensive pairing and exponentiation on limited On Board Units (OBUs) is a hard requirement to meet because of the short response time. This fact limits the VANETS in practice. The work [13] employs identity based group signature with the batch verification, provides a scalable management of large VANETs and an efficient revocation of members, but suffers from more expensive signing and verification phases than GS.

In [14], vehicles locally generate on the fly short-lived certificates (pseudonyms) with the help of GS. A Certification Authority (CA) maintains the mapping between identities and pseudonyms. One of the drawbacks is the security overhead of messages that consists of the message signature by private short-lived key, public short-lived key and the group signature of public short-lived key. In [15], the solution called TACK uses short-lived keys (ECDSA) to secure V2V messages. Long-term pre-distributed keys (group signatures) are used for anonymous authentication in regions and to gain the new certified temporary key from the Regional Authorities (RAs). TACK supports desirable short-term linkability but in dense V2I communication leads to delay in join phase and OBU must broadcast ECDSA public key with the certificate in V2V. In [10], the two proposals called SPECS include the pseudonyms maintained by Trusted Authorities (TAs), the group signature with 2-pairing batch verification and the positive and negative bloom filter for the effectiveness of the verification phase. Nevertheless, SPECS strongly rely on TAs and Road-Side Units RSUs. Also, the communication delay plays a critical role between TAs and vehicles. In [16] the authors present a Threshold Anonymous Announcement (TAA) service based on the adaptation and amalgamation of direct anonymous attestation and one-time anonymous authentication. The computational cost of the signing algorithm takes only 6 scalar multiplications and 1 pairing operation, and the computational cost of the verification algorithm takes 5 scalar multiplications and 5 pairing operations. Nevertheless, the TAA scheme does not support batch verification.

2.2 Our Contribution

Similarly like in [9], [11], [12] and [17], our proposed solution is based on group signature. We focus on the efficiency of signing/verification, security and privacy protection with respect to computationally limited RSUs. As related works, we assume OBUs with enough computational power for basic modular arithmetic, pairing and cryptographic operations.

- In the V2V communication, our solution provides the efficient signing with short-term linkability. Our proposal uses the modified scheme of Wei et al. (WLZ scheme) [17]. Nevertheless, our solution adds the short-term linkability obtaining a more efficient signing phase than in the WLZ scheme. Moreover, the WLZ scheme is focused on the V2V communication and does not describe the registration and join phases in detail. Finally, the short-term linkability is demanded for several applications [15] and can protect against Sybil and Denial of Services attacks.
- In the V2V communication, our solution provides the efficient categorized batch verification with short-term linkability. Generally in group signatures, the batch verification of n messages is more efficient than individual verification but the complexity of batch computation with bogus messages increases from O(1) to O(ln n). In [3], the authors claim that if $\geq 15\%$ of the signatures are invalid, then batch verification is not more efficient than individual verification. Our proposal modifies the WLZ scheme [17] where the batch verification costs only 2 pairings and $11n$ exponentiations. But the WLZ scheme and related solutions use uncategorized batch verification which can cause less efficient verification if bogus messages appear during attacks like the Sybil attack, the Denial of Services (DoS) attack etc. However, our solution applies categorized batch verification which sorts potential honest messages to the first batch, and potential untrusted messages to the second or third batch with lower priorities so the verification phase can be more efficient and protect against Sybil and DoS attacks.
- In V2I communication, our scheme uses probabilistic cryptography for keeping long-term unlinkability and the privacy protection of drivers. The join or registration phase takes only two messages (request/response) and the scheme does not need tamper-proof devices.
- We avoid the inefficient linear growth of revocation list with the secret keys of members. Our proposal uses the revocation process with the expiration of time stamp in certified pseudonym which revokes members by self. Vehicles have no work with a Revocation List (RL). The proposal uses only a Group Temporary Revocation List (GTRL) broadcasted between group managers to deny malicious members accessing the group of VANET members.

3 Preliminaries

In this section, we outline the scheme, the requirements and the main cryptographic techniques used in our proposal.

3.1 Scheme Description

Our scheme, depicted in Fig.2, consists of a Trusted Authority (TA), a Group Manager (GM) and a Member (V).

- **TA** issues certified member pseudonyms and generates all public cryptographic parameters in our solution. TA is fully trusted entity in our model and can reveal the real ID of a member in the revocation phase. TA is connected with all group managers and manages the registration of all members.
- **GM** is an entity which manages several Road Side Units (RSUs) and generates group secret keys to members in the join phase. In our proposal, we assume that GM is honest and is securely connected with the own RSUs (e.g. via Transport Layer Security). GM also can trace and open the malicious messages in its own area but GM cannot reveal the member ID.
- **V** is a driver with the certified pseudonym which is embedded in vehicle's OBU. After the registration of the driver in TA and joining in GM's area through the V2I communication, V can send or broadcast messages through the V2V communication. Further, V can report a bogus message through the V2I communication to GM.

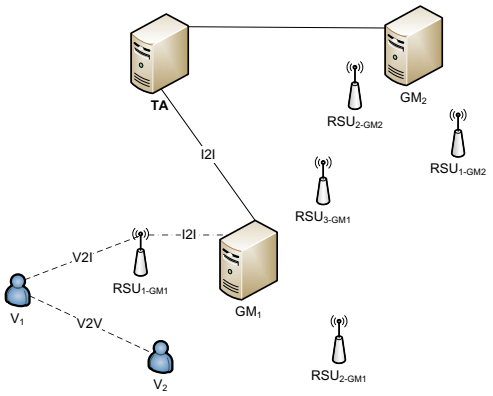


Fig. 2. The parties in our model of secure and anonymous VANET

3.2 Requirements

Our scheme is designed to satisfy these security and practical requirements:

- **Privacy (Revocable Anonymity).** Our scheme protects driver's privacy in the long-term. An honest driver with OBU can use the pseudonym signed by TA to obtain group parameters and keys from GM. Then, its OBU can sign every message on behalf of the group members and keep driver's anonymity. Every malicious driver can be revealed by the collaboration of GM and TA. If some member breaks the rules, his/her messages can be opened by GM and his pseudonym is sent to TA which can extract the member's ID. Next time, when an adversary requests a new pseudonym with a fresh time stamp (e.g. via IETF RFC 3161), TA checks if his/her ID is in the list of globally revoked members.

- **Message Integrity, Authenticity and Non-Repudiation.** In V2V communication, the group signature ensures that message is signed by a vehicle which holds the right and fresh group key pair (authenticity). The system must verify the received messages, i.e., the messages that have not been modified once they have been sent (integrity). Members stay private but can not deny that they created the signed messages (non-repudiation).
- **Short-Term Linkability.** In several VANET's applications like the safe changing of road lanes and the short-term mapping of vehicle movements, the short-term linkability is a desirable property [15]. In a short period, i.e., every 100÷300 ms, the broadcasted V2V beacon messages are used to trace vehicle's position and direction. The current proposals which use group signatures cannot link related messages from one vehicle sent in a short interval. Our scheme balances the privacy of drivers and the linkability of messages which is available only for a short interval. On the other hand, long-term unlinkability is ensured using the probabilistic encryption and changing the pseudonyms in the group signature.
- **Traceability.** When a member misuses the VANETs for his/her own benefit, he/she breaks the rules or causes an accident, the GM obtains his/her pseudonym from his/her signed messages and, sends it to the TA, who revokes the anonymity, and obtains his/her ID.

3.3 Cryptography Background

Our solution employs the ECDSA signature scheme with the public/private keys of TA, GM, V. Additionally, we use a probabilistic ElGamal encryption/decryption during the join of members. The modified short group signature WLZ scheme [17] based on the BBS04 scheme [8] is used in the V2V communication. This scheme uses bilinear maps and is based on q-SDH problem and Decision Linear problem which have been studied in [8].

We follow the notation of [8] for the concept of bilinear maps: G_1, G_2 and G_T are multiplicative cyclic groups of a prime order p. Then, g_1 is a generator of G_1, g_2 is a generator of G_2 and ψ is an isomorphism from G_2 to G_1 that $\psi(g_2) = g_1$. So e is a computable bilinear map $e : G_1 \times G_2 \rightarrow G_T$ with following properties:

- Bilinearity: for all $u \in G_1, v \in G_2$ and $a, b \in Z_p, e(u^a, v^b) = e(u, v)^{ab}$.
- Non-degeneracy: $e(g_1, g_2) \neq 1_{G_T}$.

The q-Strong Diffie-Hellman problem is a hard computational problem where $(q+2)$-tuple $(g_1, g_2, g_2^{\gamma}, g_2^{\gamma^2}, ..., g_2^{\gamma^q})$ is the input and a pair $(g_1^{\frac{1}{x+\gamma}}, x)$ is the output.

The Decision Linear Diffie-Hellman problem. Given $u, v, h, u^a, v^b, h^c \in G_1$ as input and output *yes* if $a + b = c$ and *no* otherwise, detailed in [8].

4 Our Solution

We focus on the practical registration and join of VANET members and the efficient signing/verification of V2V messages. Our solution consists of seven phases: Setup, Registration, Join, Signing, Categorized Verification, Trace, Revocation.

4.1 Setup

In the first part, TA chooses parameters $(G_1, G_2, g_1, g_2, \psi, e)$ and generates an ECDSA key pair sig_{TA}/ver_{TA}, an ElGamal private key sk_{TA} and a public key pk_{TA}, then releases the public keys and parameters. GMs generate group signature keys, ElGamal private sk_{GM_i} and public pk_{GM_i} keys for the secure V2I communication and publish public keys. Every GM_i randomly selects $r_1, r_2 \in Z_p^*, h \in G_1^*$ and sets u, v such that $u^{r_1} = v^{r_2} = h$. Then, GM_i selects random $\gamma \in Z_p^*$ and computes $w = g_2^{\gamma}$. The group public key is $gpk = (g_1, g_2, u, v, w, h)$ and the manager group secret key is $gmsk = (r_1, r_2)$.

4.2 Registration

In the registration phase, the i-th driver (member) V_i using a vehicle with OBU requests a valid certified pseudonym π_{V_i} from TA. For the first time, TA must physically verify driver's real ID, his/her driving license and OBU's ID number. Then V_i creates an ECDSA key pair sig_{V_i}/ver_{V_i}, gives the public key to TA which stores (ID_{V_i}, ver_{V_i}) in the database, and the signed certificate $cer_{V_i} = sig_{TA}(ID_{V_i}, ver_{V_i})$ is given to V_i. After the first successful registration phase, driver can request his/her next pseudonym online. Assuming that V_i has pk_{TA}, ver_{TA}, the two-message of the registration phase consists of these steps:

1. V_i self-generates ElGamal key pair (sk_{V_i}/pk_{V_i}) and sends the encrypted request $enc_{pk_{TA}}(pk_{V_i}||ID_{V_i}||ver_{V_i}||cer_{V_i}||sig_{V_i}(pk_{V_i}||ver_{V_i}||\ ID_{V_i}))$ to TA.
2. TA decrypts the request and checks if the ID_{V_i} is not revoked in Global Revocation List (GRL), the certificate cer_{V_i} and the member's signature which ensures member's authenticity and commits the pk_{V_i} in the certificate with new ElGamal key pair. Then, TA generates a challenge $c \xleftarrow{R} Z_q$, a time stamp T_l and sends the encrypted response

$enc_{pk_{V_i}}(enc_{pk_{TA}}(ID||ver_{V_i}||c)\ ||T_l||sig_{TA}(T_l||enc_{pk_{TA}}(ID||ver_{V_i}||c)||pk_{V_i}))$ back to V_i. Finally, V_i checks the signature by TA and composes the pseudonym $\pi_{V_i} = pk_{V_i}||enc_{pk_{TA}}(ID||ver_{V_i}||c)||T_l||\quad sig_{TA}(T_l||enc_{pk_{TA}}(ID||ver_{V_i}||c)||pk_{V_i})$ and stores it.

4.3 Join

A vehicle entering the i-th GM_i area (several RSUs) for the first time, requests the group public key and his/her group member secret key. We assume that RSUs managed by GM_i are securely connected through the VANET infrastructure. Let $H()$ be a hash function, and the two-message join phase consists of these steps:

1. V_i sends $\pi_{V_i} = pk_{V_i}||enc_{pk_{TA}}(ID||ver_{V_i}||c)||T_l||sig_{TA}(T_l||enc_{pk_{TA}}(ID||ver_{V_i}||c)||pk_{V_i})$,which is encrypted using pk_{GM_i}, to GM_i.
2. GM_i decrypts π_{V_i} using sk_{GM_i}, verifies π_{V_i} that is signed by TA and controls if $enc_{pk_{TA}}(ID||ver_{V_i}||c)$ is not in Group Temporary Revocation List

(GTRL) and the validity of the time stamp T_l. If π_{V_i} is ok, GM creates $gsk_{V_i} = (x_i, A_i)$, where $x_i = H(enc_{pk_{TA}}(ID||ver_{V_i}||c)||T_l||\gamma), A_i = g_1^{\frac{1}{x_i+\gamma}}$, and stores $(enc_{pk_{TA}}(ID||ver_{V_i}||c), A_i, T_l))$ to the join table and sends gsk_{V_i} encrypted using pk_{V_i} to V_i.

We note that ElGamal encryption/decryption is probabilistic. Due to this fact, an observer can not link two or more encrypted messages if V_i requests gsk_{V_i} for the second time.

4.4 Signing

The signing phase applies the modified short group signature WLZ scheme [17] which is based on the BBS04 scheme [8]. We include a counter k in the OBUs, a member secret key $gsk_{V_i} = (A_i, x_i)$ and a group public key $gpk = (g_1, g_2, h, u, v, w)$. OBU signs a message $M \in (0,1)^*$ and outputs the signature of knowledge $\sigma = (T_1, T_2, T_3, R_2, R_3, R_5, c, s_\alpha, s_\beta, s_x, s_\delta, s_\mu)$.

If $k = 0$, V_i generates $\alpha, \beta, r_\alpha, r_\beta, r_x, r_\delta, r_\mu \in Z_p^*$, and computes

$$\begin{aligned} T_1 = u^\alpha, T_2 = v^\beta, T_3 = A_i h^{\alpha+\beta}, \\ \delta = \alpha x, \mu = \beta x. \end{aligned} \tag{1}$$

$$p_1 = e(T_3, g_2), p_2 = e(h, w), p_3 = e(h, g_2). \tag{2}$$

stores $T_1, T_2, T_3, \delta, \mu, p_1, p_2, p_3$, and computes

$$\begin{aligned} R_1 = u^{r_\alpha}, R_2 = v^{r_\beta}, R_3 = p_1^{r_x} \cdot p_2^{-r_\alpha - r_\beta} \cdot p_3^{-r_\delta - r_\mu}, \\ R_4 = T_1^{r_x} u^{-r_\delta}, R_5 = T_2^{r_x} v^{-r_\mu}, \end{aligned} \tag{3}$$

$$c = H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5), \tag{4}$$

$$\begin{aligned} s_\alpha = r_\alpha + c\alpha, s_\beta = r_\beta + c\beta, s_x = r_x + cx, \\ s_\delta = r_\delta + c\delta, s_\mu = r_\mu + c\mu. \end{aligned} \tag{5}$$

Finally, V_i sends the message M with the signature $\sigma = (T_1, T_2, T_3, R_2, R_3, R_5, c, s_\alpha, s_\beta, s_x, s_\delta, s_\mu)$ and increases the counter $k{+}{+}$.

If α and β are unchanged every n messages, the short-term linkability is kept because the pseudonyms of group signature T_1, T_2, T_3 are also unchanged. Thus, for n messages, when $1 \leq k \leq n$, V_i does not need to compute equations 1, 2, contrary the WLZ scheme, but only generates random $r_\alpha, r_\beta, r_x, r_\delta, r_\mu \in Z_p^*$ and computes equations 3, 4 and 5. This reduces all 3 bilinear operations to 0, 10 exponentiations to 9 and 14 multiplications to 9. The concrete VANET application can decide when to fix the counter $k = 0$ and V_i generates new α and β and recomputes the equations 1 and 2.

4.5 Categorized Verification

Our solution uses a categorized verification which sorts the incoming signed messages to three levels of credibility. Due to the short-term linkability, V_i can keep the Temporary List (TL) of known vehicles. Firstly, the received message M_j is checked by V_i if it contains a valid time stamp, real and consistent data. After that, the message with the group signature containing T_3 is checked if T_3 is in TL. If yes, the recorded T_3 with previous validity (W=1) is included and sorted in the first batch. The validity W can be a boolean value which indicates valid (W=1) or invalid (and unknown, W=0) signatures. If no, the signed message with unknown T_3 is sorted to the second batch which is verified after the first batch verification. The rest of signed messages with T_3 linked with W=0 is verified in the third batch at the end of verification if OBU has enough time for this. This approach improves the efficiency of the batch verification process and helps when an attacker, who is out of the group, generates unsigned or corrupted messages.

Batch Verification. Batch verification is investigated in [3], and it verifies n messages in one batch. V_i uses $gpk = (g_1, g_2, h, u, v, w)$ to verify messages $\sigma_j = (T_{j1}, T_{j2}, T_{j3}, R_{j2}, R_{j3}, R_{j5}, c_j, s_{j\alpha}, s_{j\beta}, s_{jx}, s_{j\delta}, s_{j\mu})$ for $j = 1, ..., n$ does:

restores $\overline{R}_{j1} = u^{s_{j\alpha}} T_{j1}^{-c}, \overline{R}_{j4} = u^{-s_{j\delta}} T_{j1}^{s_x}$, computes a new control hash c'_j from received parameters:

$$c'_j = H(M_j, T_{j1}, T_{j2}, T_{j3}, \overline{R}_{j1}, R_{j2}, R_{j3}, \overline{R}_{j4}, R_{j5}).$$

and checks if $c'_j = c_j$. If yes then V_i continues with verification, otherwise, the message with the signature is inconsistent and is refused.

V_i randomly selects $\theta_1, \theta_2, ..., \theta_n \in Z_p$ with l_b bit,

checks batch if

$$\prod_{j=1}^{j=n} R_{j3}^{\theta_j} = e(\prod_{j=1}^{j=n} (T_{j3}^{s_{jx}} h^{-s_{j\delta} - s_{j\mu}} g_1^{-c_j})^{\theta_j}, g_2) \; e(\prod_{j=1}^{j=n} (T_{j3}^{c_j} h^{-s_{j\alpha} - s_{j\beta}})^{\theta_j}, w) \tag{6}$$

and if

$$1_{G_1} = (R_{j5} R_{j2})^{-\theta_j} T_{j2}^{\theta_j s_{jx} - \theta_j c_j} v^{(s_{j\beta} - s_{j\mu})\theta_j}. \tag{7}$$

The signed message is valid if equations 6 and 7 hold. All T_3s from new valid signed messages are added to TL with W=1. In case that the batch verification fails, the divide-and-conquer approach is used to identify the invalid signatures that were added to TL with W=0. The honest messages keep the mark W=1.

Individual Verification. In the end of the divide-and-conquer approach, the final two messages are individually verified.

V_i restores $\overline{R}_1 = u^{s_\alpha} T_1^{-c}, \overline{R}_4 = u^{-s_\delta} T_1^{s_x}$, computes new control hash c' from received parameters:

$$c' = H(M, T_1, T_2, T_3, \overline{R}_1, R_2, R_3, \overline{R}_4, R_5).$$

and checks if $c' = c$. If yes then V_i continues with the verification, otherwise, the message is inconsistent and it is refused.

Then, V_i checks if

$$R_3 = e(T_3, g_2)^{s_x} e(h, w)^{(-s_\alpha - s_\beta)} e(h, g_2)^{(-s_\delta - s_\mu)} (e(T_3, w) e(g_1, g_2)^{-1})^c \tag{8}$$

and

$$1_{G_1} = (R_5 R_2)^{-1} T_2^{s_x - cx} v^{(s_\beta - s_\mu)}. \tag{9}$$

The signed message is valid if equations 8 and 9 hold.

We can see from equations 6 and 8 that individual verification have a cost of 5 pairing operations per one message but batch verification costs only 2 pairing operations per n messages. This is the main reason why we avoid individual verification and propose to use the categorized batch verification.

4.6 Trace

Every bogus signed message can be opened by GM_i using the group manager secret key $gmsk = (r_1, r_2)$. GM_i extracts the part of the member secret group key $gsk_{V_i} \rightarrow A_i = T_3/(T_1^{r_1} \cdot T_2^{r_2})$ and searches the record $(enc_{pk_{TA}}(ID||ver_{V_i}||c), A_i, T_l)$ in the database. The part of the member pseudonym can be sent to TA for revocation.

4.7 Revocation

When there are serious circumstances, e.g., an accident, a malicious member is revoked globally by the cooperation of GM_i and TA. GM_i is able to open a message and extract the member pseudonym that is sent to TA. TA broadcasts $rev = (enc_{pk_{TA}}(ID||ver_{V_i}||c), T_l)||sig_{TA}(rev)$ to other active GMs which check the signature and store rev to own GTRLs until the lifetime of this pseudonym expire. TA extracts ID_{V_i} and adds it to GRL so the malicious member can not refresh his/her pseudonym in the next registration phase.

5 Evaluation and Security Consideration

In this section, we outline the evaluation of our solution, the comparison of the signing and verification phases with the related works which are based on group signatures and the security consideration of our solution.

5.1 Performance and Comparison with Related Work

We compare our proposal based on the BBS04 scheme [8] with the related VANETs schemes which use group signatures, the scheme of Wei et al. (WLZ scheme) [17], GSIS [7], Zhang et al. [11] and Ferrara et al. [3]. In our comparison, we omit the WS2010 scheme [12] due to the the problem in message signing which is pointed out in [17]. The verification of the TAA scheme [16] takes 5

scalar multiplications and 5 pairing operations but the TAA scheme does not support batch verification.

Generally, the time of bilinear pairing T_p is considered the most expensive operation (tens times more expensive than exponentiation operation T_e) and exponentiation is more expensive than multiplication T_m. Nevertheless, the actual processing time also depends on the input size to those operations. Due to the fact that related works are also based on the BBS04 scheme [8] we assume the same lengths of parameters (the MNT curves with $G_1 = 176$ bits, $G_T = 528$ bits and $Z_p = 162$ bits). The work [18] shows that the modular arithmetic operations like addition and subtraction can be computed more efficiently than multiplication and exponentiation. Due to this fact, we omit these fast operations in this performance evaluation.

Table 1. The comparison of the verification phases

V2V scheme:	Our scheme & WLZ scheme[17]	GSIS [7]	Zhang et al. [11]	Ferrara et al. [3]
Batch:	yes	no	yes	yes
Length of signature:	$5G_1, G_T, 5Z_p$ (2380 bits)	$3G_1, 6Z_p$ (1500 bits)	$7G_1, G_T, 5Z_p$ (2570 bits)	$3G_1, G_T, 6Z_p$ (2032 bits)
Performance of batch verification				
Pairings	**2**	5n	2	2
Exponentiation	**11n**	12n	14n	13n
Multiplication	11n+1	**8n**	17n	10n+1
Performance of individual verification				
Pairings	5	5	5	5
Exponentiation	**10**	12	12	12
Multiplication	9	8	8	8

The proposal based on the group signature BBS04 scheme [8] and motivated by Wei et al. (WLZ) [17] reaches more efficient batch verification ($2\ T_p + 11n\ T_e$), where n is the number of messages, and individual verification ($5\ T_p + 10\ T_e$) than the compared schemes, see Table 1. But the related solutions like Zhang et al. [11], Ferrara et al. [3], the WS2010 scheme [12] and also the WLZ scheme [17] use uncategorized batch verification that can be negatively affected by malicious and bogus messages ($\geq 15\%$ from all messages). To our best knowledge, our proposal applies categorized batch verification with short-term linkability in VANET for the first time. Our categorized batch verification with the temporary list of known vehicles reaches the high correctness of the important first batch in case when the bogus or damaged signed messages appear in the V2V communication.

As we can see in Table 2, our proposal significantly improves the performance of the signing of x messages with short-term linkability and it costs less operations than in the signing phase of the WLZ scheme. Pairing ($3 \Rightarrow 0$), exponentiations ($10 \Rightarrow 9$) and multiplication ($14 \Rightarrow 9$) operations are reduced.

Table 2. The comparison of the signing phases

V2V scheme:	Our scheme	WLZ scheme [17]	GSIS [7] & Zhang et al. [11] & Ferrara et al. [3]
Short-term linkability:	yes	no	no
Performance of signing for the first message / the next messages			
Pairings	3 / **0**	3 / 3	3 / 3
Exponentiation	12 / **9**	10 / 10	12 / 12
Multiplication	12 / **9**	14 / 14	12 / 12

Our scheme is implemented as a proof of concept in JAVA and uses the Java Pairing Based Cryptography (jPBC) Library [1]. The implementation employs MNT curves type D with the embedding degree $k = 6$, 171 b order curve and pre-generated parameters d840347-175-161.param and is tested on a machine with Intel(R) Xeon(R) CPU X3440 @ 2.53GHz, 4 GB Ram, Windows 7 Professional. The signing phase of our scheme with short linkability takes approx. 60 ms per one signature and the signing phase of the related schemes [3], [7], [11] and [17] based on BBS scheme takes approx. 160 ms per one signature. The verification of a single signature takes approx. 207 ms using our scheme and approx. 224 ms using related schemes. If the batch verification is employed then the verification of one signature takes approx. 50 ms so the batch verification of 10 signatures takes approx 500 ms.

5.2 Security Consideration

In this section, we outline the security consideration of our solution, that is based on the cryptographic primitives which are secure and widely accepted.

Proposition 1. *In the registration phase between V_i and honest TA, the scheme preserves message confidentiality, integrity and authenticity.*

Claim 1. *The request and response messages are confidential.*

Proof. We suppose that breaking the security of the ElGamal encryption is at least as hard as the decision Diffie-Hellman problem, as is proven in [19]. Then, the registration phase keeps confidential communication between V_i and TA due to the encryption every message by $enc_{pk_{TA}}$ and $enc_{pk_{V_i}}$. Only holder of the ElGamal private key sk_{TA} respectively sk_{V_i} can decrypt the message.

Claim 2. *The request message is authentic and can not be modified by an unauthorized entity.*

Proof. Message integrity and authenticity are ensured by the ECDSA signature scheme. The request message is unforgeable due to the commitment of the member public key pk_{V_i} in the member's certificate and in the signed part of request by V_i using ECDSA signature key sig_{V_i}. Assumining that the ECDSA signature

[1] (Available on `http://gas.dia.unisa.it/projects/jpbc/index.html`).

scheme is secure under the Elliptic Curve Discrete Logarithm Problem (ECDLP) and the used hash function is preimage resistant and collision resistant, then, if the request message was modified, the verification by the stored ECDSA key ver_{V_i} of the signature would be incorrect.

Claim 3. *The creation of a fraudulent pseudonym is computationally unfeasible nowadays.*

Proof. If an unathorized entity wants to create a pseudonym π_{V_i}, he/she needs the ECDSA private key sig_{TA} of TA. Supposing that ECDSA is secure nowadays, only trusted TA with its private ECDSA key sig_{TA} can sign π_{V_i}. Moreover, if a fraudulent π_{V_i} was sent to V_i having TA's public ECDSA key ver_{TA} then the signature of π_{V_i} would be invalid.

Proposition 2. *In the join phase between members (V_i) and honest Group Managers GM_i the proposed scheme preserves message confidentiality, integrity, authenticity and member's privacy.*

Claim 4. *The request and response messages are confidential.*

Proof. Every V_i who wants to join a group maintained by GM_i, must send the ciphertext (pk_{V_i} and π_{V_i}) encrypted using the certified ElGamal public key pk_{GM_i} to GM_i. GM_i decrypts and checks if π_{V_i} is valid and sends gsk_{V_i} encrypted using pk_{V_i}. Only V_i knows the corresponding ElGamal private key and can decrypt the message with gsk_{V_i}. Assuming that GM_i is honest, the members joining keeps the message confidentiality, integrity and authenticity due to the ElGamal properties.

Claim 5. *The pseudonym π_{V_i} is anonymous.*

Proof. Assuming that ElGamal encryption/decryption is probabilistic, an observer is unable to link two or more request/response messages because ciphertexts are different although π_{V_i} is used several times. The pseudonym π_{V_i} created by TA does not contain the plaintext of the user identity (ID) but it contains the encrypted fragment $enc_{pk_{TA}}(ID)$. GM_i and other entities without the private ElGamal key sk_{TA} are not able to open the member's ID. Hence, the privacy protection of members is ensured in the join phase.

Proposition 3. *In the V2V communication between V_i, the proposed scheme ensures message integrity, authenticity, member's privacy and revocation.*

Claim 6. *Group signatures of messages keep integrity, authenticity and non-repudiation.*

Proof. The signing and verification phases employ the group signature with the short-term linkability to ensure the message authenticity and integrity, the driver anonymity in long-term way and non-repudiation. Our scheme modifies the WLZ scheme [17] based on the BBS04 scheme [8] and inherits all its security features including the correctness. Besides honest GM_i, only a valid group member V_i can sign a message on behalf the group. If an attacker without valid $gsk_{V_i} = (A_i, x_i)$ tries to modify the message, he/she must recompute hash c

and some signature parts. Assuming that hash function is secure and the Discrete Logarithm problem holds then the computation of the proof of knowledge $(s_{j\alpha}, s_{j\beta}, s_{jx}, s_{j\delta}, s_{j\mu})$ without x_i is unfeasible nowadays. If the proof of knowledge $(s_{j\alpha}, s_{j\beta}, s_{jx}, s_{j\delta}, s_{j\mu})$ is incorrectly computed then the equations 6, 7 and 8, 6 would be not equal. The complete formal analysis can be in [8].

Claim 7. *The drivers are anonymous, untraceable by the all entities besides honest TA and their anonymities is revocable by the collaboration GM and TA.*

Proof. The group signatures contain the group members' pseudonyms T_1, T_2, T_3 which are a linear encryption of members' secret key A_i and random α and β. The short-term linkability of messages does not violate the drivers' privacy. When the counter k is set to 0 and V_i generates a new α an β then, the new signatures start to be unlinkable with old ones because contain new pseudonyms T_1, T_2, T_3. Supposing the Strong Diffie-Hellman assumption holds, every correct message of a malicious member can be opened only by GM with $gmsk = (r_1, r_2)$, and $gsk_{V_i} = (A_i, x_i)$ can be extracted. Malicious members can be revoked by the collaboration of TA and GM.

Proposition 4. *The proposed signature scheme protects against DoS attacks, Sybil attacks and replay attacks.*

Claim 8. *The categorized verification protects against DoS and Sybil attacks.*

Proof. If a malicious driver Eve (E) starts the Sybil attack which is a special kind of the DoS attack then she broadcasts bogus messages that contain fake pseudonyms and signatures. Meanwhile, the honest drivers (C, D, F,...) send messages that contain valid pseudonyms and signatures announcing an accident (sent by D) or a traffic jam (sent by C). If existing solutions are used, E can flood the uncategorized batch verification process and paralyze drivers who must discard some messages. Our proposal implements categorized batch verification. Driver Bob (B) has a Temporary List (TL) of honest drivers. We suppose that Bob's TL keeps the list of known and honest drivers like D, F,... using the property of short-term linkability which keeps the pseudonym T_3 unchanged for a short time. If B receives all messages, he checks the TL and collects the messages containing known T_3 to the first batch and verifies them. Therefore, the warning message referencing the accident from driver D is verified in time. The messages with unknown pseudonyms like C are collected to the second batch. The potentially untrusted messages from E with validity W=0 are verified in the third batch only if Bob's OBU has free time and computational capacity for this. If Eve tries to replay recent a valid pseudonyms together with false signatures then the recomputed hash c'_j is not equal to received hash c_j due to time stamps in messages. For this reason, Eve is not able to mount a successful DoS attack against the batch verification of signatures.

Claim 9. *The proposed signature scheme protects against replay attacks.*

Proof. Every message M contains besides position speed etc. also a time stamp with actual time and date. Before verification, every received message is checked

if the time stamp is actual and valid. If an attacker without valid $gsk_{V_i} = (A_i, x_i)$ wants replies an old message with valid signature of a user, he/she must modify the time stamp to valid and actual one, then, recomputes hash c_j, and recomputes all parts $s_{j\alpha}, s_{j\beta}, s_{jx}, s_{j\delta}, s_{j\mu}$ of the signature. Anyway, recomputing valid $s_{jx}, s_{j\delta}, s_{j\mu}$ without x_i is unfeasible under the Discrete Logarithm problem.

6 Conclusions

In this paper, we have presented our anonymous solution using short-term linkable group signature with categorized batch verification. Our proposed solution deals with anonymous and secure signing/verification of messages in the V2V communication which is more efficient than related works. Further, the solution provides practical and secure registration, join and revocation of members in VANETs. The short-term linkability significantly improves the signing phase and is demanded in several VANET applications. Our categorized batch verification provides less errors in the important first batch of potentially honest messages. Moreover, the categorized batch verification protects against Sybil and DoS attacks. Our future plans are aimed at the investigation of categorized batch verification and short-term linkability in dense traffic. The variable values, e.g., the size of counter k affecting short-term linkability, will be determined for various traffic scenarios.

Acknowledgments. This work was partially supported by the Technology Agency of the Czech Republic project TA02011260; the Ministry of Industry and Trade of the Czech Republic project FR-TI4/647; the Spanish Ministry of Science and Innovation (through projects eAEGIS TSI2007-65406-C03-01, CO-PRIVACY TIN2011-27076-C03-01, ICTW TIN2012-32757, ARES-CONSOLIDER INGENIO 2010 CSD2007-00004 and Audit Transparency Voting Process IPT-430000-2010-31), by the Spanish Ministry of Industry, Commerce and Tourism (through projects eVerification2 TSI-020100-2011-39 and SeCloud TSI-020302-2010-153) and by the Government of Catalonia (under grant 2009 SGR 1135).

References

1. Sun, J., Fang, Y., Zhu, X.: Privacy and emergency response in e-healthcare leveraging wireless body sensor networks. Wireless Com. 17(1), 66–73 (2010)
2. Hussain, R., Kim, S., Oh, H.: Towards Privacy Aware Pseudonymless Strategy for Avoiding Profile Generation in VANET. In: Youm, H.Y., Yung, M. (eds.) WISA 2009. LNCS, vol. 5932, pp. 268–280. Springer, Heidelberg (2009)
3. Ferrara, A.L., Green, M., Hohenberger, S., Pedersen, M.Ø.: Practical Short Signature Batch Verification. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 309–324. Springer, Heidelberg (2009)
4. Gerlach, M., Festag, A., Leinmuller, T., Goldacker, G., Harsch, C.: Security architecture for vehicular communication. In: The 5th International Workshop on Intelligent Transportation (March 2007)

5. Fonseca, E., Festag, A., Baldessari, R., Aguiar, R.: Support of anonymity in VANETs - putting pseudonymity into practice. In: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC), Hong Kong (March 2007)
6. Raya, M., Hubaux, J.P.: Securing vehicular ad hoc networks. J. Comput. Secur. 15, 39–68 (2007)
7. Lin, X., Sun, X., Han Ho, P., Shen, X.: GSIS: A secure and privacy preserving protocol for vehicular communications. IEEE Transactions on Vehicular Technology 56, 3442–3456 (2007)
8. Boneh, D., Boyen, X., Shacham, H.: Short Group Signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
9. Zhang, C., Lu, R., Lin, X., Ho, P.H., Shen, X.: An efficient identity-based batch verification scheme for vehicular sensor networks. In: INFOCOM, pp. 246–250. IEEE (2008)
10. Chim, T.W., Yiu, S.M., Hui, L.C.K., Li, V.O.K.: SPECS: Secure and privacy enhancing communications schemes for VANETs. Ad Hoc Networks 9(2), 189–203 (2011)
11. Zhang, L., Wu, Q., Solanas, A., Domingo-Ferrer, J.: A scalable robust authentication protocol for secure vehicular communications. IEEE Transactions on Vehicular Technology 59(4), 1606–1617 (2010)
12. Wasef, A., Shen, X.S.: Efficient group signature scheme supporting batch verification for securing vehicular networks. In: IEEE International Conference on Communications, ICC (2010)
13. Qin, B., Wu, Q., Domingo-Ferrer, J., Zhang, L.: Preserving Security and Privacy in Large-Scale VANETs. In: Qing, S., Susilo, W., Wang, G., Liu, D. (eds.) ICICS 2011. LNCS, vol. 7043, pp. 121–135. Springer, Heidelberg (2011)
14. Calandriello, G., Papadimitratos, P., Hubaux, J.-P., Lioy, A.: Efficient and robust pseudonymous authentication in VANET. In: Proceedings of the Fourth ACM International Workshop on Vehicular Ad Hoc Networks, VANET 2007, pp. 19–28. ACM, New York (2007)
15. Studer, A., Shi, E., Bai, F., Perrig, A.: Tacking together efficient authentication, revocation, and privacy in VANETs. In: SECON, pp. 1–9. IEEE (2009)
16. Chen, L., Ng, S.L., Wang, G.: Threshold anonymous announcement in VANETs. IEEE Journal on Selected Areas in Communications 29(3), 605–615 (2011)
17. Wei, L., Liu, J., Zhu, T.: On a group signature scheme supporting batch verification for vehicular networks. In: International Conference on Multimedia Information Networking and Security, pp. 436–440. IEEE C. S., Los Alamitos (2011)
18. Malina, L., Hajny, J.: Accelerated modular arithmetic for low-performance devices. In: The 34th International Conference on Telecommunications and Signal Processing (TSP), pp. 131–135 (August 2011)
19. Tsiounis, Y., Yung, M.: On the Security of ElGamal Based Encryption. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 117–134. Springer, Heidelberg (1998)

GHUMVEE: Efficient, Effective, and Flexible Replication

Stijn Volckaert*, Bjorn De Sutter, Tim De Baets, and Koen De Bosschere

Computer Systems Lab, Ghent University
{stijn.volckaert,bjorn.desutter,koen.debosschere}@elis.ugent.be

Abstract. We present GHUMVEE, a multi-variant execution engine for software intrusion detection. GHUMVEE transparently executes and monitors diversified replicae of processes to thwart attacks relying on a predictable, single data layout. Unlike existing tools, GHUMVEE's interventions in the process' execution are not limited to system call invocations. Because of that design decision, GHUMVEE can handle complex, multi-threaded real-life programs that display non-deterministic behavior as a result of non-deterministic thread scheduling and as a result of pointer-value dependent behavior. This capability is demonstrated on GUI programs from the Gnome and KDE desktop environments.

Keywords: Memory Exploits, Non-determinism, Diversified Process Replicae.

1 Introduction

Memory error exploits divert the control [2] or data flow [10] of a vulnerable program by injecting faulty data. This is typically done by overwriting data such as code pointers. Examples of such exploits are stack-smashing [2], return-oriented programming [30], and return-to-lib(c) attacks [27]. Such attacks nearly always rely on knowledge about the memory layout of the attacked application.

Several protection strategies exist to fix the vulnerabilities [5, 40], to protect against buffer overflows at run-time [1, 12, 44] to protect against the execution of injected code [22], and to prevent the attacker from determining the addresses of data [29]. Modern OSes and system libraries support all of these approaches to *prevent intrusions* and to *prevent damage* in case of intrusions. For example, the Linux and glibc support Address Space Layout Randomization (ASLR) [29] and Exec Shield [26] to prevent code on the stack to be executed, and length-bounded string functions like `strncpy` and `strncat` [38]. Extensions have been proposed in academics, such as Address Space Layout Permutation (ASLP) [20].

Many protections have been circumvented, however. Return-to-lib(c) [27] and return-oriented programming [30] attacks simply do not require injected code to be executed, the use of more secure library functionality like `strncpy` has proven to be error-prone [25] and ASLR was attacked in a brute-force manner [37].

* The authors want to thank the Agency for Innovation by Science and Technology in Flanders (IWT) and Ghent University for their support.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 261–277, 2013.

A more reliable protection based on *intrusion detection* was proposed in 2006. Cox et al [14] implemented a Linux kernel extension to transparently run multiple diversified replicae of the same application in parallel. The protection relies on the assumption that it is much harder for an attacker to make diversified replicae perform the exact same malicious behavior than it is to exploit vulnerabilities in a single application version. The replicae are executed in lock-step and are always transparently fed the exact same input. A monitor module compares the invoked output operations of the replicae before executing them. When the monitor detects any discrepancies, it assumes that those result from an attack taking place and it terminates the execution before any damage is done.

Since Cox et al, a number of other so-called multi-variant execution engines (MVEE) have been developed [8, 9, 32–36], as well as different methods to diversify applications, including stacks growing in opposing directions [32], heap layout randomization [6], redundant data diversity [28], address space partitioning [8], ASLR [29, 37], and code diversification [3, 4, 41].

However, a major problem of all existing MVEEs is that they cannot handle real desktop applications. The fundamental reason is that real-world applications are not deterministic because of non-deterministic thread scheduling and because their behavior depends on concrete pointer values, which vary when replicae are diversified. By contrast, pre-existing MVEEs and their diversification schemes only function on simple, single-threaded applications. Moreover, their memory layout diversification is limited to relatively weak, predictable forms.

This paper presents the Ghent University MVEE or GHUMVEE. Contrary to the existing MVEEs, GHUMVEE's design supports a wide range of features observed in non-deterministic applications and a wider range of diversification techniques, including the stronger protection of the less predictable, full ASLR. Furthermore, experiments demonstrate that GHUMVEE comes with less performance overhead than existing MVEEs. The most fundamental novel aspect of GHUMVEE's design is its ability to intervene in the execution of replicae at program points other than system calls. This does require some cooperation of the application developer, but as we will discuss in the paper, compilers can easily limit the burden on the developer.

Section 2 discusses related work and the weaknesses of existing MVEE's that GHUMVEE's design overcomes. Section 3 presents this design, which is evaluated in Section 4. Section 5 draws conclusions.

2 Related Work

Software memory exploit techniques and countermeasures have been actively researched in the past 15 years. Stack overflow attacks have long been the easiest way to seize control of a running application. Smashing [2] the stack allows for an attacker to inject shell code or overwrite return addresses [27]. Several solutions were proposed to eliminate stack overflow attacks. StackGuard [12] inserts canaries into the stack to detect return address overwrites. Several state-of-the-art compilers adapted this technique later on [19, 24]. Other people proposed

the use of a secondary stack to keep copies of the return addresses [11, 21]. An alternative is to extend the C-library functions that are commonly used to set up the attack, with extra security checks such as bounds checking [1, 44]. LibSafe [5, 40] does this at runtime and modern versions of the GCC and VC++ compilers offer alternative versions of these functions [16, 23]. Mainstream operating systems also implement some form of software-enforced Data Execution Prevention [22, 26] to prevent injected code from being executed. Most other exploiting techniques, control-data attacks in particular, share one important property. They all make assumptions about the memory layout of the target application, e.g., about the absolute locations of certain functions or about the distance between two allocated objects. Many proposed techniques attempt to break these assumptions, all of which involve randomization. Xu [43] modified the Linux program loader to dynamically relocate a program's stack, heap and shared libraries. The PaX team implemented and demonstrated Address Space Layout Randomization [29], a well known technique with goals similar to the latter. ASLR employs a kernel patch to relocate a program's stack, heap and shared libraries during startup. All mainstream operating systems have adapted ASLR. Other techniques exist but are not as commonly used [13, 20].

In 2005, Berger and Zorn proposed DieHard [6], a framework for redundant execution of multiple diversified program replicae. DieHard tries to protect programs against memory errors and exploits thereof by running multiple replicae of the same program in parallel and feeding them the same input. A custom memory allocator ensures full heap randomization of the replicae: objects always have different addresses in the different replicae. DieHard redirects all program output through `stdout` to its voter module where it can isolate replicae that encounter memory errors. DieHard does not require any kernel modifications but can only run replicae whose only input comes from `stdin`, it cannot run multi-threaded programs or any other programs with pointer-dependent behavior.

More advanced MVEEs are the N-Variant Systems [14, 28], the proof-of-concept MVEE from Cavallaro et al [8, 9] (hereafter called CPoC), and Orchestra [32–36]. Figure 1 displays their basic operation. The kernel or user-space monitor is responsible for running multiple replicae of a process in a user-transparent manner. To that extent, the monitor intercepts all communication between the replicae and the outside world. As the progress of the replicae (denoted by black bars on the horizontal time access) may differ, they will communicate through system calls at different moments in time. The monitor intercepts the calls, stalls the calling replicae and waits until all of them have made a call. At that time, the monitor compares the calls and operands, and either terminates the program or handles the calls appropriately. For example, when `sys_brk` is invoked to request memory from the OS, the monitor checks the requested sizes and lets the OS allocate memory for both replicae after which both replicae continue executing. When `sys_write` is invoked to write to a file, the monitor blocks one of them to ensure transparency for the user. The result of the system call is still fed back to both replicae, which continue their execution.

The first one concerns the *rendez-vous points* at which MVEEs intervene in the execution of the replicated program versions. The aforementioned MVEEs only intervene in a replica to intercept system calls. By construction, those MVEEs can therefore not handle any non-trivial multi-threaded program. The reason is that the threads in the different replicae have to be executed in the same order (i.e., synchronized) to avoid false alerts. No fully deterministic execution is required (i.e., consecutive runs of the application under control of an MVEE may feature different thread schedules), but within one execution under control of an MVEE, all synchronization events and decisions need to be replicated. For example, when a program allocates tasks in a pool to spawn task threads, they need to be spawned in the same order in all replicae. And when the tasks updates shared memory, that needs to happen in the same order in all replicae. MVEEs that only intercept applications upon system calls cannot provide this synchronization for two reasons. First, many modern applications feature synchronization operations that do not involve system calls. These include atomic functions, (uncontended) locks by means of futexes [17], and many custom synchronization primitives. Secondly, several synchronization primitives such as the pthread_cond_timedwait execute multiple system calls as part of a more abstract decision process. To replicate these decisions, a replication mechanism at a higher abstraction level than system calls is needed as well. As will be discussed in detail in Section 3.3, GHUMVEE supports such a replication mechanism that solves both issues with acceptable performance overhead.

The second limitation of existing MVEEs is their lack of support for program behavior that depends on exact pointer values. N-Variant Systems and CPoC rely on address space partitioning, in which each concrete address that can be targeted by an attack occurs in only one replica. Orchestra features two replicae in which the stack grows in different directions to prevent a buffer overflow from having the same effect in both replicae.

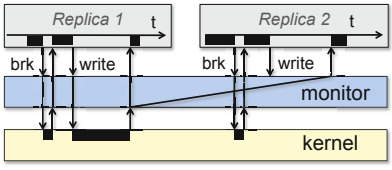


Fig. 1. Basic operation of a MVEE

The problem with these address-space based approaches is that many applications' behavior depends on concrete pointer values. Those values are then typically hashed to index hash tables or other containers such as (supposedly unordered) sets. When the computed hashes differ in the different replicae, their behavior diverges in many ways. For example, when collusions in a hash table differ in two replica, they might rehash or resize the hash tables at different points in time. In the case of a resize, this might involve memory allocation system calls being executed in one replica but not in the other. None of the existing MVEEs can handle this. When iterating over supposedly unordered containers in which objects are stored based on hashed pointer values, the order will also depend on the concrete values. So the order of visiting objects and performing tasks on them might differ from one replica to the other. In some cases the tasks involve no sensitive operations, but in many cases they do. This ranges from

different files being opened for different objects, over locks being taken on the visited objects, to worker threads being spawned for the stored objects.

In other words, if the order in which objects are stored in containers is not controlled by the MVEE, different replicae may show diverging behavior in every possible way. All existing MVEEs that we are aware of suffer from this problem. This has two consequences. First, they are applicable only to relatively simple, nice programs. For example, the programs evaluated in the existing MVEE papers are limited to a modified Apache, thttpd, SPEC benchmarks, and Snort. Exactly how nice the programs have to be depends on the precise details of the MVEE internals. Secondly, because of this dependence on different aspects of nice behavior, the existing MVEEs provide only relatively weak forms of protection. Orchestra is limited to protecting buffer overflows on the stack. The partitioned address spaces of N-Variant Systems and CPoC ore a very limited form of layout diversification with very predictable behavior.

As discussed in Section 3.7, GHUMVEE can handle many modern programs with address-dependent behavior, none of which are handled correctly by pre-existing MVEEs. Moreover, GHUMVEE can replicate the applications with full code and data layout randomization. As such, GHUMVEE makes the protection provided by replicated execution applicable to a much wider set of applications, and it demonstrates that stronger forms of protection can be provided. These are the two most important contributions of this paper.

3 GHUMVEE Architecture

The GHUMVEE monitor is launched from the command line with the program to be protected as its argument. From a database GHUMVEE then retrieves the executables of the replicae. GHUMVEE spawns the replica processes to which it attaches itself using Linux' `ptrace` API [39]. From then on, GHUMVEE acts as a proxy between the replicae and the kernel as depicted in Figure 1.

3.1 Rendez-Vous Points

GHUMVEE can intercept all system calls invoked by the replicae and manipulate or stall them when needed. GHUMVEE's rendez-vous points are system call entries (i.e, invocations) and exits (i.e., returns). Replicae are stalled at both types of points and not resumed until all replicae have reached the rendez-vous point. GHUMVEE handles rendez-vous points based on the type of system call the replicae are trying to execute. We generally distinguish four types of system calls. The distinction is based on four factors:

I/O-Related System Calls: These system calls should be performed only once to ensure transparency and to avoid unwanted side effects. For example, when replicae are writing to a file, the data should be written only once, precisely like it would happen in the original program.

Side Effects: System calls that create, modify or delete process-bound kernel structures have side effects. Most memory management functions are examples of such system calls. These calls are performed by all replicae in the same manner as depicted in Figure 1.

Mutable Results: System calls that have mutable results, i.e., calls that return different results upon every invocation, should only be performed once to ensure that all replicae get consistent return data from these calls. Most time-related functions are examples of such calls.

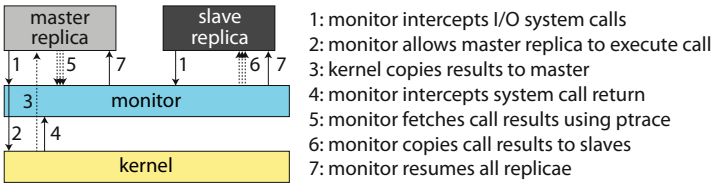
Self-Aware: System calls that make a process self-aware should only be performed once. These include `sys_getpid` and `sys_open(/proc/self/...)`.

After a system call entrance has been handled in accordance with the call's class, the monitor waits for the replicae to hit the next rendez-vous point. In most cases, this is the system call's exit. Handling this rendez-vous point is straightforward. If the system call was executed by all replicae, the monitor checks whether all of them received consistent results from the call. Then it either resumes them or shuts down the system, e.g., if a call returned an error for some replica but not for the others. If on the other hand, the call was only executed by one replica, the monitor copies the return data into the address spaces of the slave replicae, after which it resumes all of them.

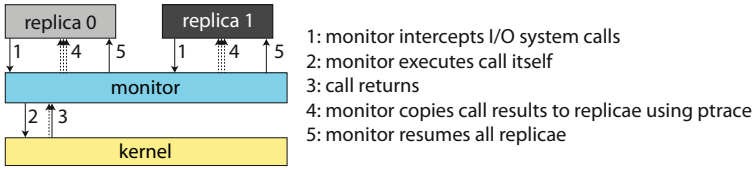
3.2 I/O Replication and Data Transfers

As mentioned above, MVEEs generally allow I/O related system calls to be executed only once. Nearly every existing solution deals with this restriction differently. Cox et al [14] stall all slave replicae in kernel-space while the master replica executes the actual I/O call. When that call returns, the monitor copies its return value and return data to the address spaces of the slave replicae.

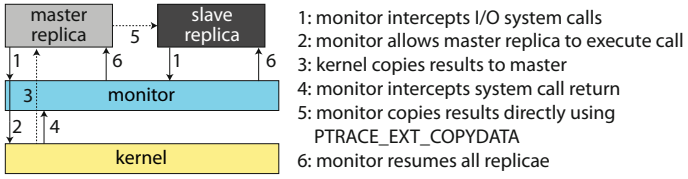
Later MVEEs handle I/O replication in user-space using Linux debugging facilities such as the `ptrace` and `waitpid` APIs [39]. The `ptrace` API allows for a debugger to observe and control the execution of a debuggee process by inspecting and manipulating its internal state, while `waitpid` is used to poll a debuggee for status changes such as the entrance or exit of a system call. CPoC's [9] implementation is similar to Cox'. The fact that CPoC stalls the replicae in user-space does entail an additional issue, however. When a process (e.g., a slave replica) is stalled at the entrance of a system call, that process cannot be prevented from executing the actual call once it is resumed. To skip such as system call, a debugger has to replace its number in register EAX by that of another system call that has no side effects. The best choice for this purpose is `sys_getpid`. After replacing the system call number and resuming the replaced slave calls, CPoC waits until the master replica returns from the original system call and until the slaves returns from their fake `sys_getpid` calls. CPoC then first copies the results of the master system call from from master replica to the monitor, and then from the monitor to all slave replicae. This process is visualized in Figure 2(a), in which solid arrows denote control transfers and dashed arrows denote data being copied.



(a) CPoC's I/O replication



(b) Orchestra's basic I/O replication



(c) GHUMVEE's I/O replication

Fig. 2. I/O replication in three MVEEs

Salamat [34] implemented a different system in Orchestra. Rather than letting a master replica execute a system call, Orchestra executes the call on behalf of the variants and copies the results of the call directly from the monitor to the replicae. This is visualized in Figure 2(b).

GHUMVEE's implementation of I/O replication is nearly identical to CPoC's. Like CPoC, we only allow the master replica to execute the original system call. Unlike CPoC however, we often do not copy the results of the system call from the master to the monitor. Instead, we copy the results directly from the master to the slaves as shown in Figure 2(b). As a result, GHUMVEE performs one less memory copy operation per replicated I/O call than CPoC and Orchestra.

In Figure 2, the copying between monitor and replicae is depicted with multiple arrows. This reflects the limitation of copying only one memory word at a time with the `PTRACE_PEEKDATA` and `PTRACE_POKEDATA` operations. On the x86 architecture, this implies that at most 4 bytes can be copied per peek or poke, each of which requires the monitor to perform a `ptrace` system call. Even in the simplest applications, this introduces a significant performance penalty.

Salamat [34] proposed a workaround that consists of shared memory buffers between the monitor and the replicae, the standard `memcpy` to copy data between that shared memory and the monitor's private memory, and a custom `memcpy`

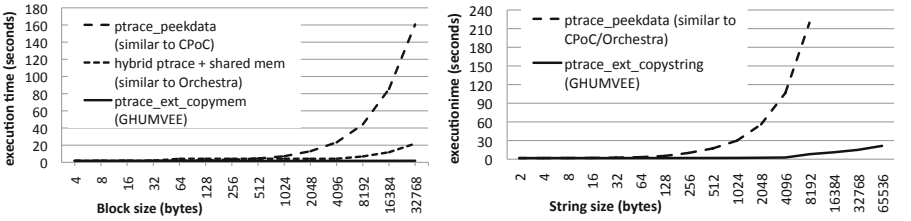


Fig. 3. Comparison of different data transfer methods

function injected into each replica. For every transfer of 40 bytes or more, control in the replicae is diverted to the injected functions to transfer data from the shared buffers to the replicae's private memories. So every transfer requires two copies: one into the shared memory and one out of it. A similar overhead would exist when `/proc/<pid>/mem` would be used instead, as that cannot be mapped directly into a process' address space.

GHUMVEE avoids part of this overhead with two small extensions for the `ptrace` API in Linux [39]. The `PTRACE_EXT_COPYDATA` and `PTRACE_EXT_COPYSTRING` operations enable a monitor or a debugger to copy a fixed-size data block and a NULL-terminated string directly to, from, and between any of its replicae or debuggees. GHUMVEE uses these extensions for all data transfer operations when it finds them in the kernel, hence the horizontal arrows in Figure 2(c). On synthetic performance benchmarks that stress the data transfer functionality of our MVEE, we obtained the results depicted in Figure 3. This figure shows that GHUMVEE's optional kernel extensions allow for data to be transferred much more efficiently. In real-world benchmarks such as SPEC CPU 2006, these extensions improved multi-variant performance by 1 to 4%.

3.3 Multi-threading and Synchronization

Arguably the biggest challenge for a MVEE is to deal with multi-threaded replicae. This is complicated mainly because MVEEs running in user-space cannot control the order in which threads are scheduled[1]. This implies (1) that a monitor can observe system calls in different orders in multi-threaded replicae because of different progress rates and different scheduling of the replicae, and (2) that the replicae of programs with non-deterministic behavior can actually perform different system calls in different replicae.

The first problem can be solved easily with a multi-threaded monitor. Like the other MVEEs that support fork/exec and multi-threading, the GHUMVEE monitor spawns a new monitor thread for every set of new processes or threads spawned by the replicae. This works fine as long as the replicae behave deterministically and execute in lock-step, because then they will spawn the same processes and threads from within the same processes and threads. Each new

[1] And even when the monitor would run in kernel-space, if has no direct control over user-space synchronization events, so the fundamental problems remain the same.

monitor thread then attaches to the corresponding replicae threads, after which each such monitor thread observes only the system calls in those corresponding threads, which will happen in exactly the same order in all replicae.

The second problem is much harder to solve. Pre-existing MVEEs simply neglect this and are hence broken for many applications, for which they report mismatches between the replicae and halt the execution. Fundamentally, the problem is that any synchronization race in non-deterministic programs can lead to different replicae executing different system calls in different orders.

In GHUMVEE, we solved this problem by forcing all slave replicae to behave exactly like the master. Whenever a synchronization race in the master replica is decided, that same decision is imposed onto the slave. This is similar to techniques used for record/replay of multi-threaded applications [31], the difference being that in GHUMVEE all replicae run concurrently in lock step, rather than sequentially. We therefore don't have to store logs of the synchronization events. Please note that GHUMVEE does not eliminate non-determinism. Rather, it only forces all replicae to take the same decision for every synchronization race. This way, GHUMVEE cannot introduce any deadlocks in the replicae.

Initially, we interposed [15] or detoured [18] all user-space synchronization operations by means of fake system calls through which the monitor became aware of the operations for which it could then enforce scheduling decisions. This solution introduced too much overhead, however. Even simple applications like the `gcalctool` calculator from the Gnome desktop environment spawn several threads during their initialization, in which they perform mostly uncontended synchronization. For example, we observed `gcalctool` performing 1.8M futex operations during its 400 ms initialization. Interposing all those operations with a fake system call and multiple ptrace system calls made the initialization time grow to over 370 seconds, a slowdown with a factor 925!

As an alternative, we designed a system with which the replicae can synchronize themselves. This is visualized in Figure 4. When the monitor spawns the replicae, it allocates shared circular buffers (shown in green) between them. Furthermore, the monitor preloads a dynamic library with interposers and detours to intervene in all user-space synchronization events. Instead of executing system calls in all replicae as in our initial solution, these new interposers record the synchronization decisions of the master replica (e.g., the order in which its threads acquire locks) in the shared buffers. In the figure, the master threads record the order in which they acquired a specific lock Lm in the buffer. In the slave replicae, the interposers read these decisions, and impose the same behavior on the slaves. In the figure, when slave thread Bs first tries to acquire the corresponding lock Ls, the interposer observes that thread As should acquire it first, so it blocks thread Bs. When thread As tries to acquire the lock, this succeeds, and after it is released, thread Bs will acquire it as well.

As all interposers perform their duties without additional system calls or context switches to the monitor, the overhead of this solution is much smaller. For example, with this solution the already mentioned `gcalctool` initializes in 1.7 seconds, a slowdown with factor 4.25. This is still significant, but most of it

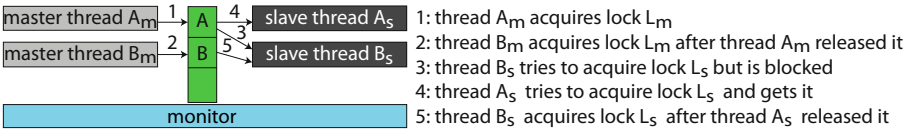


Fig. 4. GHUMVEE's synchronization decisions through shared buffers

is due to setting up the shared buffers. After the initialization the rate at which synchronization operations are performed decreases significantly, as a result of which this enforced synchronization between different replicae does not results in an noticeable overhead during the normal, interactive operation of the program.

Enforcing the master's schedule on the slave replicae in this way solves non-determinism problems related to synchronization races, a form of non-determinism that is generally considered acceptable program behavior and that occurs in most modern multi-threaded programs. Our solution does not protect against non-determinism caused by critical data races. As such data races are generally considered as bugs, we feel this is an acceptable limitation of GHUMVEE.

3.4 Signal Handling

Besides control-data dependencies in multi-threaded applications, there are several other sources of non-determinism. One of them is asynchronous signal delivery. Because most signal handlers invoke system calls, delivering signals from external sources to replicae should happen very carefully. For example, assume that one single-threaded replica is blocked on entry to a system call, waiting for the other single-threaded replicae to arrive at the same point. If we then deliver a signal to another replica that is still executing, the corresponding signal handler in that replica will be invoked, in which a very different system call might be invoked in turn, leaving two replicae wanting to execute different system calls.

In GHUMVEE, this is solved by delaying the delivery of signals to replicae until they are blocked on exit of a system call. This can significantly delay the handling of a signal. Salamat et al proposed a complex solution to deliver signals earlier [34], which showed significant improvements for synthetic signal handling stress tests. We investigated the need for such a complex solution for real-world applications, and discovered that in real applications, the number of signals is typically more than three orders of magnitude lower than the number of system calls invoked. As such, limiting the delivery of signals to the rendezvous points of those system calls does not hinder performance or latency in practice. One notable side effect of our signal handling mechanism is that some duplicate asynchronous signals might be lost. In practice however, we have not encountered any programs that started behaving incorrectly when a duplicate signal was not delivered.

Unlike asynchronous signals, synchronous signals are delivered immediately. Synchronous signals occur as a direct result of the executing instruction. Because the MVEE keeps all replicae in a consistent state, we can assume that all replicae will trigger the same synchronous signal on the same instruction.

3.5 Time Stamp Counter

Yet another source of non-determinism is time, of which applications can become aware through the `gettimeofday` system call. On the x86 architecture, the time can also obtained directly from the processor by executing the `rdtsc` instruction in user mode. Salamat [34] acknowledge the problems this can cause when different replicae get different input through `rdtsc`, but he offers no solution beyond pointing out that programmers could use `gettimeofday` instead. In practice, programmers do not follow his advice, however. A simple program like the `gcalctool` calculator executes the `rdtsc` instruction tens of times.

GHUMVEE solves this by setting the control registers in the x86 architecture to make all user-mode `rdtsc` instructions trap. The monitor handles the resulting `SIGSEGV` signal by feeding all replicae the same time stamp counter value.

3.6 Shared Memory Support

Linux programs can use shared memory blocks to set up communication channels with other programs. Once such channels are in place, the programs can communicate by reading and writing from and to the shared memory without using any system calls and without exchanging any other information. This makes it very hard for a MVEE to perform correct replication under all circumstances.

Somewhat surprisingly, almost all programs use shared memory. So at least partial support is needed in an MVEE. But fortunately, most programs do not really require two-way communication channels with the outside world via shared memory. An analysis of the usage of shared memory in our testing applications reveals that shared memory is typically used for one of the following goals:

Shared Libraries: The Linux program loader uses shared memory blocks to map shared libraries into the address spaces of dynamically linked programs. This should obviously be supported by an MVEE.

Memory-Mapped I/O: When a file is mapped into a program's address space as shared memory, it can be read and written without the overhead of system calls. We have encountered several programs in the KDE desktop environments that require memory-mapped I/O to start up properly.

Internal Communication: Anonymous shared memory is not accessible to external programs. We have encountered several multi-threaded programs that used anonymous shared memory to set up additional heaps, e.g., with large contiguous pages. Anonymous shared memory can only be accessed by the allocating program and its descendants. Since these descendants also run under MVEE control, all communication through anonymous shared memory can be controlled using the techniques described in Section 3.3.

Non-anonymous 2-Way Communication Channels: Several programs try to set up 2-way communication with the outside world through the System V `sys_ipc` system call. As indicated, this cannot be handled efficiently. We also discovered, however, that all programs we studied have backup schemes for when the System V call is not supported, i.e., when it

fails. That backup uses the above types of shared memory, as well as regular communication channels like pipes, signals and system calls. As those channels can be handled by MVEEs without a problem, it suffices to let the monitor intercept the requested shared memory allocation by means of the System V system calls and let them return as if the requests failed.

Several solutions have been proposed in the past to deal with the first three cases [9, 34]. GHUMVEE builds on those solutions. Although the classification above seems pretty straightforward, it is not easy to allow all safe uses of shared memory while blocking the unsafe forms. Memory-mapped I/O is particularly hard to support because memory-mapped files and regular 2-way communication channels are set up the same way. Cavallaro [9] proposed to solve this problem by using the CPU's page exception mechanism but indicated that this approach might incur a lot of overhead. For that reason we did not even consider this solution. Instead GHUMVEE supports memory-mapped I/O by manipulating the `sys_mmap` and `sys_mmap2` used to map shared memory onto files. Normally, memory-mapped files are mapped by passing the `MAP_SHARED` flag to the mmap call. Our monitor disables this flag and enables the `MAP_PRIVATE` flag instead. This way, the requested file is mapped into the address spaces of the replicae, but any changes to the file are not written back to the file when the block is unmapped. Instead the GHUMVEE monitor keeps track of these manipulated blocks and performs the write-back of the file data itself.

This approach prevents programs from using shared memory based 2-way communication channels without notifying the programs, but in practice, we have not encountered any program that stopped working because of our solution.

3.7 Address Space Layout

Finally, we observed that many real-world applications and libraries (including GTK+, Glib, Pango, KDE, and LibreOffice libraries) exhibit behavior that depends on pointer values. As explained in Section 2, the main problem with pointer values being hashed into keys to access data structures is that the data structures are resized, restructured or iterated through in orders that depend on keys obtained from hashing pointer values. As a result of these dependencies, almost all non-trivial programs we tried fail on existing MVEE's with ASLR enabled. Nonetheless, this problem is not mentioned in any MVEE-related paper.

We tackle this problem by interposing the hash functions that compute pointer-dependent keys, similarly to the way we interpose synchronization operations. In the master replica, the interposer wraps the hash function and passes the computed keys to shared queues. In the slave replicae, the interposers replace the hash functions. Instead of computing a hash key, they obtain them from the queues. That way, all replicae use the same hash keys. This solution is not fool proof, as it only works for limited uses of the hash keys, such as for indexing and ordering data structures. In more complex scenarios, e.g., where the hashing is replaced by encryption and where the encrypted keys also get decrypted, GHUMVEE can still fail. But for the applications we tested that use the aforementioned libraries, GHUMVEE works fine.

4 Experimental Evaluation

Validation. We tested GHUMVEE on numerous interactive, multi-threaded programs, including Gnome tools such as `gcalctool`, KDE tools such as `kcalc`, and LibreOffice on a quadcore Core i7 870 system running Ubuntu 11.04. For, e.g, LibreOffice we tested operations such as opening and saving files, editing various types of documents, running the spell checker, etc. In these tests GHUMVEE spawned between one and four replicae from the same executable. Tests were conducted with and without ASLR enabled. Without ASLR, all addresses occurring in the replicae are identical. With ASLR, most addresses are different in all replicae. This includes the addresses of data on the heap and on the stack, as well as addresses of statically allocated data and code in dynamically linked libraries. We also evaluated GHUMVEE on a number of SPEC benchmarks, mainly to evaluate performance and to validate GHUMVEE on replicae with code diversification. In particular, we compiled the SPEC2006 benchmark with GCC 4.5.2 at optimization levels -O2 and -O3. This allows us to test GHUMVEE on replicae in which even the static code addresses differ.

All tests succeeded. This demonstrates that GHUMVEE is more flexible than existing MVEEs, in the sense that it supports a wider range of applications, as well as a wider range of data and code diversification techniques, including full layout randomization which presents a much less predictable target to attackers.

Transparency. From a user perspective, GHUMVEE is completely transparent. Except for having to launch an application with the GHUMVEE monitor and the performance and memory consumption overhead involved with the use of GHUMVEE, there is no noticeable effect.

From a developer perspective, however, GHUMVEE is not completely transparent. In particular GHUMVEE's reliance on interposers is not fully transparent. To handle synchronization as described in Section 3.3 and address space layout differences as described in Section 3.7, someone has to implement the appropriate interposers. Besides highly application-dependent use of pointer values to index data structures, many real-world applications use custom synchronization mechanisms [42] as well as custom memory allocators [7] besides the standard `glibc` and `pthread` primitives (despite the cited literature demonstrating how bad that customization practice is). In all these cases, the application developers themselves are responsible (1) for ensuring that interposers can handle all cases correctly, and (2) for providing the interposers.

For our whole test suite, we wrote 2863 lines of interposer C code. Their distribution over different libraries is shown in Table 1. Of those 2983 lines, 2509 cover the functionality in standard libraries and the header files of the GHUMVEE interposer API. Those are readily available to all GHUMVEE users and need not be reimplemented. For specific applications (Gnome and LibreOffice) and the libraries they rely on, 474 lines of very simple C interposer code suffice. For example, defining an interposer for the hash function `gtk_gc_value_hash` that takes an argument of type `gpointer` (as defined in that library) to produce a hash of type `guint` looks as follows:

Table 1. Programming effort for GHUMVEE's interposers

standard library	interposer library (header files)		libc	pthread	interposer base lib	total
lines of C code	260		654	766	829	2509
application library	glib	gtk	orbit	pango	libreoffice	total
lines of C code	105	54	78	54	183	474

```
INTERPOSER_DETOUR_GEN_HOOKFUNC(guint, gtk_gc_value_hash, (gpointer key)) {
    MVEE_DO_SYNC(guint, (key), int, MVEE_GTK_HASH_BUFFER, 0);
    return result;
}
```

This code specifies that the slave replicae should obtain the hashed value with the width of an `int` from the master through the `MVEE_GTK_HASH_BUFFER` instead of computing a hash themselves (`0`). `INTERPOSER_DETOUR_GEN_HOOKFUNC` and `MVEE_DO_SYNC` are preprocessor macro's defined in one of the GHUMVEE source code headers. The first macro generates an empty trampoline [18] and generates detour registration code that automatically detours [18] the target function (`gtk_gc_value_hash`) when the interposer library is loaded. The second macro generates all of the synchronization code. In the master replica this synchronization code writes all computed hash values into a circular buffer that is shared between all replicae. In the slave replicae, the code reads the computed values from that same buffer.

Writing the code for other interposers is similarly mechanistic. The effort required by the developer is therefore by and large limited to identifying the functions that need to be interposed and to make them interposable. For the latter, i.e., for ensuring that interposers can handle all cases correctly, we only needed to modify 54 lines of code in the applications and their libraries to convert static and hence non-interposable functions into dynamic, non-inlined interposable ones. This only required removing the `static` keyword and `inline` attribute from the code. Additionally, the LibreOffice link-script had to be adapted to make the symbols corresponding to the hash functions visible in the linked binary. This also is a trivial edit.

More changes were needed in `glibc`, which contains a large amount of inline assembly. Besides the 654 lines of interposer code, our `glibc` patch is 845 lines long. This patch can of course be reused by all GHUMVEE users.

Altogether, this limited and mechanistic programming overhead (part of which can probably be automated by a compiler working on the basis of pragmas) demonstrates the limited burden on application programmers to make their applications compatible with GHUMVEE. Because of its reliance on widely applicable and easy to use interposers, GHUMVEE is a much more flexible tool than existing MVEEs.

Performance Overhead. To measure the performance overhead of GHUMVEE, we measured the execution times of several multi-variant combinations of SPEC

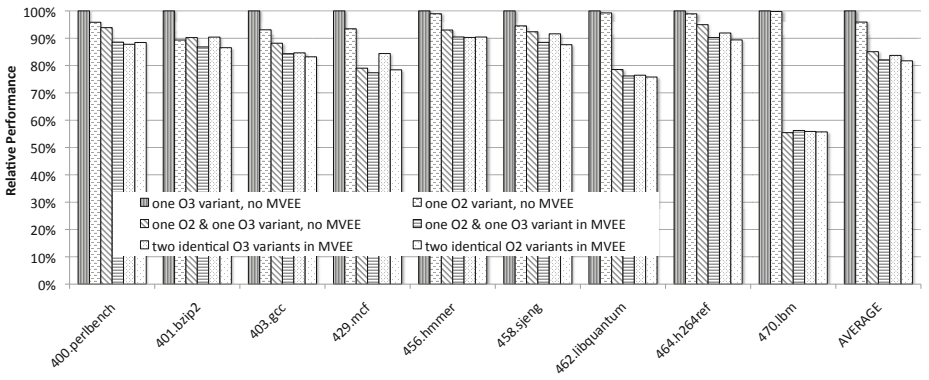


Fig. 5. Relative performance of SPEC benchmarks running under the GHUMVEE

benchmarks compiled at different optimization levels. The relative performance (i.e., the relative execution times) are depicted in Figure 5.

Comparing the results of GHUMVEE with two O3 variants (yellow bars) to those with one O3 variant (gray bars), we observe that on average GHUMVEE comes with a 16% performance penalty. For O2 variants (orange vs. blue), the performance penalty is 15%. The average overhead of GHUMVEE is hence slightly smaller than the 17% reported for Orchestra [34], despite the fact that GHUMVEE performs equivalence checks for many more system calls.

For `403.gcc`, the only benchmark common to our evaluation and that of Orchestra, the overheads are 15% and 17% for O3 and O2 resp. with GHUMVEE, and about 30% with Orchestra. Given that `403.gcc` is an I/O intensive benchmark, this difference in performance can be attributed to the kernel extension discussed in Section 3.2.

For `470.lbm`, the overhead of 46% is particularly high. This overhead is not due to the overhead of GHUMVEEs interventions, however. A similar overhead can be observed when two variants of this benchmark run side by side without an MVEE. As `470.lbm` is a memory-intensive benchmark, the shared caches and the shared memory buses on the Core i7 become the bottleneck when running multiple variants concurrently, with or without the MVEE. This demonstrates that although GHUMVEE limits the overhead of its interventions, it cannot magically reduce the inherent overhead of replicating processes that are not fit for replication due to resource contention. In this regard, GHUMVEE comes with the same limitations as any other MVEE.

5 Conclusions

We presented the Ghent University multi-variant execution engine or GHUMVEE, the first intrusion detection system based on the execution of diversified replicae that supports full ASLR and real-life applications. We presented novel techniques to support thread synchronization and pointer-dependent program behavior, as

well as other sources of non-determinism such as time stamp counters. To the extent a comparison with existing systems was possible, GHUMVEE proved to be more effective, more efficient, and more flexible than existing MVEEs.

References

1. Akritidis, P., Costa, M., et al.: Baggy bounds checking: an efficient and backwards-compatible defense against out-of-bounds errors. In: Proc. USENIX SSYM, pp. 51–66 (2009)
2. Aleph One: Smashing the stack for fun and profit. Phrack Magazine 7(49) (1996)
3. Anckaert, B.: Diversity for Software Protection. PhD thesis, Ghent University (2008)
4. Anckaert, B., Jakubowski, M., Venkatesan, R.: Proteus: virtualization for diversified tamper-resistance. In: Proc. ACM DRM, pp. 47–58 (2006)
5. Baratloo, A., Singh, N., Tsai, T.: Libsafe: Protecting critical elements of stacks. White paper, Bell Labs, Lucent Technologies (December 1999)
6. Berger, E., Zorn, B.: DieHard: probabilistic memory safety for unsafe languages. In: Proc. ACM PLDI, pp. 158–168 (2006)
7. Berger, E.D., Zorn, B.G., McKinley, K.S.: Reconsidering custom memory allocation. In: Proc. ACM OOPSLA, pp. 1–12 (2002)
8. Bruschi, D., Cavallaro, L.: Diversified Process Replicæfor Defeating Memory Error Exploits. In: Proc. IEEE IPCCC, pp. 434–441 (2007)
9. Cavallaro, L.: Comprehensive Memory Error Protection via Diversity and Taint-Tracking. PhD thesis, Universita Degli Studi Di Milano (2007)
10. Chen, S., Xu, J., Sezer, E., Gauriar, P.: Non-control-data attacks are realistic threats. In: Proc. USENIX SSYM (2005)
11. Chiueh, T.C., Hsu, F.H.: RAD: A Compile-Time Solution to Buffer Overflow Attacks. In: Proc. IEEE ICDCS, pp. 409–417 (2001)
12. Cowan, C., Pu, C., et al.: StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In: Proc. USENIX SSYM, pp. 26–29 (1998)
13. Cowan, C., Beattie, S., Johansen, J., Wagle, P.: PointGuard: Protecting Pointers from Buffer Overflow Vulnerabilities. In: Proc. USENIX SSYM, pp. 91–104 (2003)
14. Cox, B., Evans, D., et al.: N-variant systems: A secretless framework for security through diversity. In: Proc. USENIX SSYM, pp. 105–120 (2006)
15. Curry, T.W.: Profiling and Tracing Dynamic Library Usage Via Interposition. In: Proc. USENIX USTC, pp. 267–278 (1994)
16. Holtmann, M.: Secure Programming with GCC and GLibc (2008)
17. Franke, H., Russell, R., Kirkwood, M.: Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux. In: Proc. Ottowa Linux Symposium (2002)
18. Hunt, G., Brubacher, D.: Detours: Binary Interception of Win32 Functions. In: Proc. USENIX WINSYM (1999)
19. IBM Research: GCC extension for protecting applications from stack-smashing attacks (2005)
20. Kil, C., Jun, J., Bookholt, C., Xu, J., Ning, P.: Address space layout permutation (aslp): Towards fine-grained randomization of commodity software. In: Proc. ACSAC, pp. 339–348 (2006)
21. McGregor, J.P., Karig, D.K., Shi, Z., Lee, R.B.: A Processor Architecture Defense against Buffer Overflow Attacks (2003)
22. Microsoft Corporation: Data Execution Prevention

23. Microsoft Corporation: Security Enhancements in the CRT
24. Microsoft Corporation: Visual C++ Linker Options: /GS (Buffer Security Check) (2002)
25. Miller, T.C., de Raadt, T.: strlcpy and strlcat Consistent, Safe, String Copy and Concatenation. In: Proc. USENIX ATEC, pp. 175–178 (1999)
26. Molnar, I.: "Exec Shield", new Linux security feature
27. Nergal: The advanced return-into-lib(c) exploits. Phrack Magazine 12(58) (2001)
28. Nguyen-Tuong, A., Evans, D., Knight, J.C., Cox, B., Davidson, J.W.: Security through redundant data diversity. In: Proc. IEEE DSN, pp. 187–196 (2008)
29. PaX Team: Address Space Layout Randomization (2004)
30. Roemer, R., Buchanan, E., et al.: Return-oriented programming: Systems, languages, and applications. ACM Trans. Inf. Syst. Secur. 15, 2:1–2:34 (2012)
31. Ronsse, M., De Bosschere, K.: RecPlay: A Fully Integrated Practical Record/Replay System. ACM Trans. Comp. Sys. 17(2), 133–152 (1999)
32. Salamat, B., Gal, A., Franz, M.: Reverse stack execution in a multi-variant execution environment. In: CATARS Workshop (2008)
33. Salamat, B., Jackson, T., et al.: Orchestra: A User Space Multi-Variant Execution Environment. In: Proc. EuroSys, pp. 33–46 (2009)
34. Salamat, B.: Multi-Variant Execution: Run-Time Defense against Malicious Code Injection Attacks. PhD thesis, University of California, Irvine (2009)
35. Salamat, B., Gal, A., et al.: Multi-variant Program Execution: Using Multi-core Systems to Defuse Buffer-Overflow Vulnerabilities. In: Proc. CICIS, pp. 843–848 (2008)
36. Salamat, B., Jackson, T., et al.: Orchestra: intrusion detection using parallel execution and monitoring of program variants in user-space. In: Proc. EuroSys, pp. 33–46 (2009)
37. Shacham, H., Goh, E.J., Modadugu, N., Pfaff, B., Boneh, D.: On the effectiveness of address-space randomization (2004)
38. The GNU C Library: Copying and Concatenation
39. Thorvalds, L.: Linux Programmer's Manual
40. Tsai, T., Singh, N.: Libsafe 2.0: Detection of Format String Vulnerability Exploits (2001)
41. Williams, D., Hu, W., et al.: Security through Diversity: Leveraging Virtual Machine Technology. IEEE Security & Privacy 7(1), 26–33 (2009)
42. Xiong, W., Park, S., Zhang, J., Zhou, Y., Ma, Z.: Ad hoc synchronization considered harmful. In: Proc. USENIX OSDI, pp. 1–8 (2010)
43. Xu, J., Kalbarczyk, Z., Iyer, R.K.: Transparent Runtime Randomization for Security. In: Proc. SRDS 2003, pp. 260–269 (2003)
44. Younan, Y., Philippaerts, P., et al.: Paricheck: an efficient pointer arithmetic checker for C programs. In: Proc. ASIACCS, pp. 145–156 (2010)

Extracting Attack Scenarios Using Intrusion Semantics

Sherif Saad and Issa Traore

University of Victoria, BC, Canada
shsaad@ece.uvic.ca , itraore@engr.uvic.ca

Abstract. Building the attack scenario is the first step to understand an attack and extract useful attack intelligence. Existing attack scenario reconstruction approaches, however, suffer from several limitations that weaken the elicitation of the attack scenarios and decrease the quality of the generated attack scenarios. In this paper, we discuss the limitations of the existing attack scenario reconstruction approaches and propose a novel hybrid approach using semantic analysis and intrusion ontology. Our approach can reconstruct known and unknown attack scenarios and correlate alerts generated in multi-sensor IDS environment. Our experimental results show the potential of our approach and its advantages over previous approaches.

Keywords: Attack Scenario, Alerts Correlation, Intrusion Analysis, Semantic analysis.

1 Introduction

In the last several years the number of computer network attacks has rapidly increased while at the same time the attacks have become more and more complex and sophisticated. Intrusion analysts and network administrators need to understand these attacks to take appropriate responses and design adequate defensive and prevention strategies. In particular, they need to reconstruct the attack scenario (also known as attack plan) to extract attack intelligence. The attack scenario elicits the steps and actions taken by the intruder to breach the system. Understanding the attack scenario allows the intrusion analyst to identify the compromised resources, spot the system vulnerabilities, and determine the intruder objectives and the attack severity.

The current generation of intrusion detection systems (IDSs) generate low level intrusion alerts that describe individual attack events. In addition, existing IDSs tend to generate massive amount of alerts with high rate of redundant alerts and false positives. Typical IDS sensors report attacks independently and are not designed to recognize attack plans or discover multistage attack scenarios. Moreover, not all the attacks executed against the target network will be detected by the IDS. False negatives, which correspond to the attacks missed by the IDS, will either make the reconstruction of the attack scenario impossible or lead to an incomplete attack scenario. Because of the above mentioned reasons, manual

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 278–292, 2013.

reconstruction of attack scenarios is a challenging task. Hence, there is a pressing need for new techniques allowing automatic reconstruction of attack scenarios.

We propose, in this paper, a new attack scenario reconstruction technique, which improves the attack scenario reconstruction process by combining two complementary approaches: semantic-based alerts clustering and causality-based attack analysis. More specifically, an initial set of candidate attack scenarios are first identified by measuring the similarity between IDS alerts through semantic analysis. The candidate attack scenarios are then refined by analyzing the causal relationships between them using an intrusion ontology.

We evaluated experimentally our approach using two popular datasets yielding excellent performances. In the literature the ***completeness*** (also known as the true detection rate) and ***soundness*** of the alerts correlation are the most adopted metrics to evaluate attack scenario reconstruction approaches. The two metrics were proposed by Ning et al [8]. Completeness is computed as the ratio between the number of correctly correlated alerts by the number of related alerts (i.e. that belong to the same attack scenario). Soundness is defined as the ratio between the number of correctly correlated alerts by the number of correlated alerts. The completeness metric captures how well we can correlate related alerts together while the soundness metric assesses how correctly the alerts are correlated.

The experimental evaluation of our approach yielded for both datasets, soundness and completeness ranging between 96% and 100% for the sample attack scenarios considered.

The remaining of the paper is organized as follows. Section 2 summarizes and discusses previous works on attack scenario reconstruction. Section 3 introduces our semantic model and the underlying concepts and metrics. Section 4 presents in detail our attack scenario reconstruction technique. Section 5 shows the result of our experiment. Finally, in section 6 we conclude this paper and point out some future research directions.

2 Related Works

Several approaches have been proposed in the literature for attack scenario reconstruction. The proposed approaches fall into one of two main categories based on the type of data analysis techniques involved as explained below.

The first category of attack scenario reconstruction approaches use data clustering and data mining techniques, either to cluster alerts based on their attributes similarity or to mine alerts sequences in specific time interval. Under this category fall the approaches proposed by Li *et al.*, Ding *et al*, and Al-Mamory and Zhang, respectively.

Li *et al.* investigated multi-step attack scenario reconstruction using association rule mining algorithms [5]. The authors assumed that multi-step attacks often happen in a certain time interval and based on this assumption an attack sequence time window is defined and used for association rule mining. The DARPA 2000 dataset was used to evaluate the proposed approach yielding attack scenario detection rate of 92.2%.

Ding *et al.* proposed an attack scenario reconstruction model by extending the apriori association rule mining algorithm to handle the order of intrusion alerts occurrence [3]. The authors introduced, more specifically, a time sequence apriori algorithm for mining intrusion alerts with respect to their order of appearance. The DARPA 1999 dataset was used to evaluate the proposed algorithm. The evaluation results show that the true scenario detection rate is 76% while the soundness of the approach is 53%.

Al-Mamory and Zhang proposed a lightweight attack scenario reconstruction technique by correlating IDS alerts based on their statistical similarity [2]. In the proposed approach, similar raw IDS alerts are grouped into meta-alert (MA) messages. An attack scenario is generated by correlating MA messages using a relation matrix (RM) that defines the similarities between every two MA messages. Using the DARPA 2000 dataset, it was shown that the completeness and the soundness of the proposed approach are 86.5% and 100%, respectively.

Attack scenario reconstruction systems that use clustering and data-mining approaches can handle large amount of IDS alerts and in general can reconstruct novel and unknown attack scenarios. They suffer, however, from several limitations. One of these limitations is the inability of the techniques to reconstruct complex or sophisticated multi-step attack scenarios. This is because clustering and data-mining approaches cannot detect causality between individual attacks. Another important issue is their proneness to construct incorrect attack scenarios. For instance, the alert clustering process may lead to overlapping alerts clusters. Alerts from the same scenario may end up in different alerts clusters, while alerts from different scenarios may be placed in the same cluster. It is not possible, however, for one alert instance to belong to two different attack scenarios at the same time. Such situation can occur because either an alert actually belongs to one scenario and is falsely clustered into the other scenario, or there is only one real attack scenario, and the reconstruction technique falsely assumes that there are two scenarios.

The second category of approaches use, in most cases, rule bases for attack scenario reconstruction, and represent attack scenarios and attack knowledge using formal methods. Examples of works that fall under this category include proposals by Ning *et al.*, Ding, and Liu *et al.*, respectively.

Ning and colleagues proposed an attack reconstruction approach that correlates intrusion alerts based on the prerequisites and the consequences of the intrusion [8]. The intrusion prerequisites are the necessary conditions for the intrusion to occur and the intrusion consequences are the outcomes of successful intrusions. The DARPA 2000 DOS 1.0 attack scenario dataset was used to evaluate the proposed technique, yielding an equal value for the completeness and the soundness of 93.96% .

Liu and colleagues proposed a multi-step attack scenario reconstruction technique using predefined attack models [7]. The proposed technique defines attack models that an attacker may follow to break in the system. Each defined attack model follows a general attack pattern involving four phases: probe, scan, intrusion, and goal. The attack scenario reconstruction is executed over three main

stages, namely, preprocessing stage, attack graph construction stage, and scenario generation stage. The proposed technique was evaluated using the DARPA 2000 LLDOS1.0 dataset achieving 87.12% completeness and 86.27% soundness.

The above knowledge-based approaches can reconstruct both known and unknown attack scenarios as long as the individual attack steps are stored in the knowledge-base. In addition some of these approaches can capture the causality between individual attacks. However, most of the proposed systems use hard coded knowledge and rely on explicit knowledge. As a result, these techniques fail to detect hidden and implicit relations between attacks, which makes it difficult for them to recognize novel attack instances in a timely fashion. Moreover, knowledge-based techniques cannot handle concurrent attacks that do not have any explicit causal relationship.

Based on the above literature review, it is clear that new approaches are needed that can handle large amount of IDS alerts and allow reconstructing automatically novel and unknown attack scenarios with high accuracy. In this regard, we propose a new attack scenario reconstruction approach that is a hybrid of clustering and knowledge-based techniques.

To improve the accuracy of clustering-based reconstruction, a robust alert clustering criteria must be defined. Clustering IDS alerts is difficult because many alerts attributes are symbolic data, and also heterogeneous IDS sensors tend to use different formats and vocabularies to describe the alerts. To address the above challenges, we propose to cluster the alerts based on their semantics and not their syntactic representations. After clustering, we refine using semantic inference the obtained clusters by identifying causally related alerts subsets and linking such subsets to specific attack scenarios.

3 Intrusion Semantic Analysis

We use an ontology to describe the intrusion domain and encode our knowledge base. The use of an ontology involves two main advantages. Firstly, it provides a common vocabulary to describe IDS alert messages generated by different IDS sensors. This allows achieving interoperability between heterogeneous IDS sensors. Secondly, it provides a semantic representation for the domain of computer and network intrusions. Using the semantic representation of IDS alerts and intrusion instances allows analyzing the alerts and the intrusion based on their semantic characteristics, and inferring the underlying relationships.

Several network intrusion ontologies have been proposed in the literature [1,13,4]. We use in our work a new intrusion ontology, introduced in our previous work [11] that contains the required knowledge to extract intrusion intelligence.

The intrusion ontology contains many classes representing different concepts from the intrusion analysis domain. The upper level classes of our intrusion ontology are illustrated in Figure 1. Classes in the ontology are connected by arcs representing the relations between them.

The relations between concepts can be quantified by measuring their *semantic relevance*. In knowledge engineering and information retrieval, the notion of *relevance* expresses how two objects are related with respect to the matter at hand.

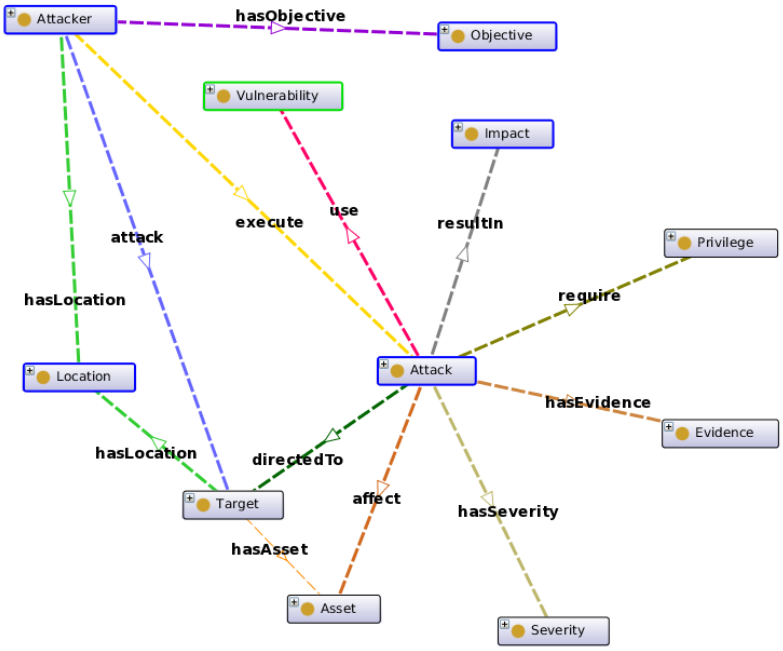


Fig. 1. Intrusion Ontology Screenshot

Semantic relevance occurs between classes and individuals in the same ontology through either explicit relations or implicit relations. Several approaches have been proposed to calculate the semantic relevance between concepts, objects or resources in specific domain of knowledge [10,9]. We propose in this work, a new metric to capture the semantic relevance between intrusion alerts based on the relations occurring between them through our ontology. More specifically, we compute the semantic relevance between two alerts x and y as the summation of the weights of all the relations occurring between them divided by the summation of the weights of all the relations that can occur between any two alerts.

Given two alerts $x \in A, y \in A$, let R_{xy} denote the set of all relations between x and y. Let R denote the set of all relations between alerts pairs from A, i.e., $R = \cup_{x \in A, y \in A} R_{xy}$. Given a relation $r \in R$, let $w(r)$ denote the weight associated with r. We define the semantic relevance between alerts x and y as follows:

$$sem_{rel}(x, y) = \frac{\sum_{r \in R_{xy}} w(r)}{cardinality(R)} \tag{1}$$

In order to compute the semantic relevance between two alerts, we need to identify their relations, both implicit and explicit. While explicit relations are drawn from predefined ontological relationships, implicit relations are discovered through semantic inference.

A subset of the ontological relations used to calculate the semantic relevance between alerts are shown in Figure 2. Using these relations, a set of inference rules were designed. The rules are represented in the Semantic Web Rule Language (SWRL) and stored as XML files in the knowledge-base. Table 1 shows some of the predicate sentences (used to define the rules) and their meanings.

Table 1. Predicates Sample

Predicate Sentence	Description
Alert(?x)	check if variable x is an Alert instance
Attack(?a)	check if variable a is an Attack instance
report(?x,?a)	check if variable a which is an attack instance is reported by x which is an alert instance
Impact(?m) ∧ resultIn(?a,?m)	check if variable a which is an attack instance has an impact m which is an instance of attack impact class

The following is an example of an inference rule that finds if two alerts have the same attacker:

$$Alert(?x) \wedge Alert(?y) \wedge Attacker(?a) \wedge hasSource(?x, ?a) \wedge hasSource(?y, ?a) \rightarrow hasSameAttacker(?x, ?y)$$

A chain of rules can be used to infer an indirect relation between two alerts. For example, it can be established by inference that two different alerts that report two different attack types while having the same impact are relevant. An example of SWRL rule to infer alerts with similar attack impact is given by:

$$Attack(?a) \wedge Attack(?b) \wedge Impact(?m) \wedge resultIn(?a, ?m) \wedge resultIn(?b, ?m) \rightarrow hasSameImpact(?a, ?b)$$

$$Alert(?x) \wedge Alert(?y) \wedge Attack(?a) \wedge Attack(?b) \wedge report(?x, ?a) \wedge report(?y, ?b) \wedge hasSameImpact(?a, ?b) \rightarrow reportSameImpact(?x, ?y)$$

4 Attack Scenario Reconstruction

4.1 General Approach

Our attack scenario reconstruction process starts by collecting raw alerts generated by different (heterogeneous or homogeneous) IDS sensors, with different formats and containing possibly some false positives. The collected raw alerts are preprocessed by converting them into a common format that takes into account both the structures and semantics of the alert messages. Then, the converted alerts are validated by eliminating possible false positives. To convert the alerts into a common format, a separate profile is built for each IDS sensor. Each sensor profile contains a set of formatting rules used to convert raw alerts into a predefined format based on the vocabularies in the intrusion ontology.

The alerts resulting from the previous phases are grouped into several clusters based on their semantic relevance. The obtained clusters are analyzed using semantic inference to detect the causality relation between corresponding alerts. Then, the attack scenarios are extracted using semantic inference.

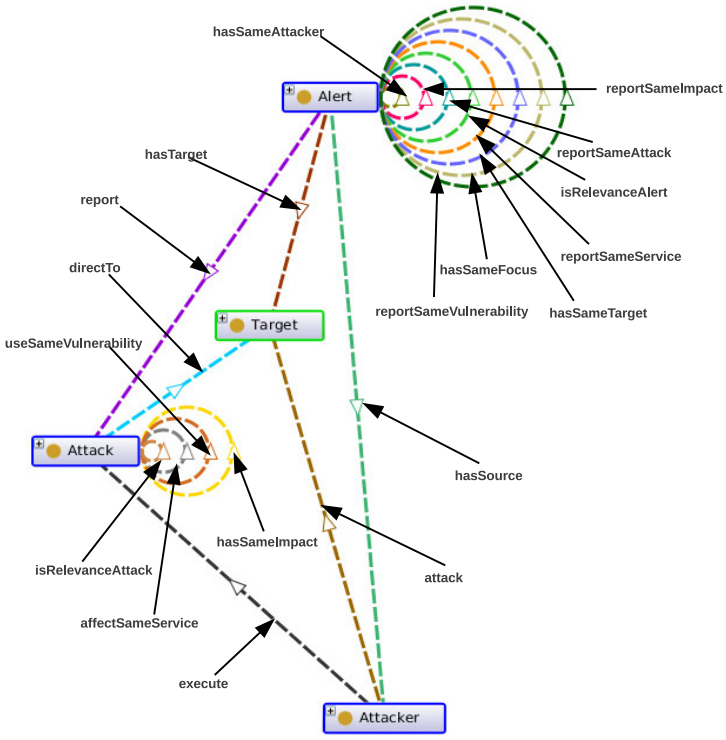


Fig. 2. Ontological Relations between Alerts, Attack, Attacker and Target

4.2 Semantic-Based Alerts Clustering

The objective of semantic-based alerts clustering is to find groups of alerts that are semantically relevant with respect to particular attack scenarios. A cluster of semantically relevant alerts represents a candidate attack scenario. Given a set A of n number of alerts there are $2^n - 1$ possible alerts groupings, where each alert grouping corresponds to a candidate attack scenario. A generated candidate attack scenario may correspond to a true or false attack scenario.

Based on the inferred relations between alerts, we calculate the semantic relevance between them and construct what we refer to as the alerts correlation graph (ACG). The ACG is an undirected weighted graph $G = (V, E)$, where V is a set of vertices representing alerts and E is a set of edges representing the relations between alerts. The edges in the ACG are labeled by the values of the semantic relevance between the alerts corresponding to adjacent vertices.

As an example, suppose we want to construct the ACG for the set of alerts given in Table 2.

For the sake of simplicity we will assume that only three types of relations can occur between any two alerts, namely, ***hasSameSource***, ***hasSameTarget*** and ***reportSameAttack***, and also that each relation has a weight value equal 1. This means that the maximum number of relations between any two alerts is

Table 2. Alerts Examples

ID	Source	Target	Attack
a_1	201.134.12.11	172.16.112.10	Scan
a_2	201.134.12.11	172.16.116.44	Scan
a_3	135.13.216.191	172.16.113.84	Scan
a_4	201.134.12.11	172.16.112.10	BufferOverFlow
a_5	135.13.216.191	172.16.116.44	Scan
a_6	201.134.12.11	172.16.112.10	RootAccess
a_7	135.13.216.191	172.16.116.44	TelnetAccess

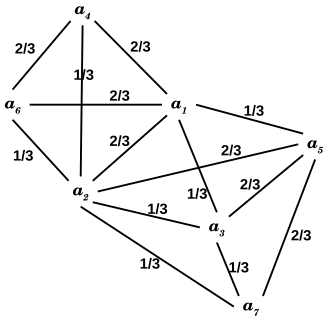


Fig. 3. Example of Alerts Correlation Graph

3. Based on the above considerations, the constructed ACG for the alerts set in Table 2 is shown in Figure 3.

The edges of the ACG in Figure 3 are labelled by the semantic relevance values between corresponding alerts. For instance, alerts a_1 and a_6 being linked by two relations (i.e. ***hasSameSource*** and ***hasSameTarget***), the semantic relevance between them is 2/3.

Algotithm 1 illustrates the steps to build the Alerts Correlation Graph. The algorithm takes a set A of hybrid or commonly formatted alerts as an input and generate the alerts correlation graph as an $n \times n$ matrix G where n is the total number of alerts in A. The entry $G[i, j]$ is zero if the semantic relevance between alerts a_i and a_j in A is less than a predefined semantic relevance threshold θ. If the semantic relevance value w is greater than or equal θ the algorithm set the value of $G[i, j]$ equal to w, which indicates that there is an edge e between a_i and a_j in G with weight w. The runtime complexity of Algorithm 1 is $O(n^2)$.

In graph theory a clique in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge. In our case a clique in the ACG represents a subset of semantically relevant alerts. Therefore, we consider every maximum clique in the ACG as a candidate attack scenario. We use the well-known Bron-Kerbosch algorithm to find all maximum cliques in the ACG. In the ACG shown in Figure 3, there are three maximum cliques as illustrated by Figure 4.

Algorithm 1. Constructing Alerts Correlation Graph

```
   /* A a set of IDS alerts                                          */
   /* G a matrix represent the ACG                                   */
   /* w a semantic relevance between a pair of alerts in A           */
   /* θ semantic relevance threshold                                 */
   /* n number of alerts in A                                        */
   Input: A, θ
   Output: G
 1 begin
 2   for i ← 1 to n − 1 do
 3     for j ← i + 1 to n do
 4       w ← sem_rel(a_i, a_j);
 5       if w ≥ θ then
 6         G[i, j] ← w ;
 7       end
 8     end
 9   end
10   return G;
11 end
```

Now let c_1, c_2, and c_3 denote the three maximum cliques in the ACG of Figure 4, where $c_1 = \{a_1, a_2, a_4, a_6\}$, $c_2 = \{a_1, a_2, a_3, a_5\}$ and $c_3 = \{a_2, a_3, a_5, a_7\}$. By looking closely at the above three candidate attack scenarios, we notice that they have some common vertices (alerts). For example, a_2 belong to all three of them. Considering that an alert can belong to only one attack scenario, we need to refine our set of candidate attack scenarios by removing common alerts between them.

To remove a common alert from different candidate attack scenarios, we calculate the total semantic relevance of the common alert with respect to each candidate attack scenario, and assign it to the candidate attack scenario yielding the maximum total semantic relevance. This process will be repeated until each alert is assigned to only one candidate attack scenario.

The total semantic relevance of an alert with respect to a specific attack scenario is the sum of the semantic relevance between this alert and other alerts in the same attack scenario. For example, in Figure 4 the total semantic relevance of vertex a_1 in c_1 is $(2/3 + 2/3 + 2/3 = 2)$ and in c_2 is $(2/3 + 1/3 + 1/3 = 1.3)$. Therefore, a_1 will be removed from c_2 and reassign to only c_1. By applying the same method to other common vertices, we will end up with only two candidate attack scenarios s_1 and s_2, where $s_1 = \{a_1, a_2, a_4, a_6\}$ and $s_2 = \{a_3, a_5, a_7\}$.

Algorithm 2 illustrates the main steps to extract the candidate attack scenarios from an alert correlation graph. The algorithm takes as input an alert correlation graph G generated by Algorithm 1. First the set C of maximum cliques are extracted from G using the Bron-Kerbosch algorithm. The alerts (or vertices) in each clique are sorted based on the alert number. To detect alerts that belong to more than one clique we apply a simple set intersection method, where each clique in C is treated as a set. The set intersection returns a list

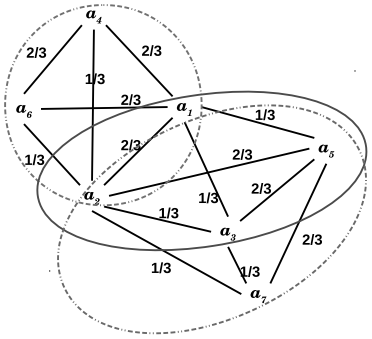


Fig. 4. Maximum Cliques in an Alerts Correlation Graph

$A^{'}$ of alerts (vertices) that belong to more than one clique. Then, the algorithm iterates for n times, where n is the total number of alerts in $A^{'}$. In each iteration the algorithm calculates the alert membership to each clique in C based on the total semantic relevance. At the end of each iteration an alert a is assigned to a clique c, where the membership of a with c is maximum. Then, a is removed from the other cliques in C. Finally the algorithm removes a from $A^{'}$ and terminates when $A^{'}$ is empty. In addition to extracting candidate attack scenarios, Algorithm 2 addresses also the problem of shared alerts between the candidate scenarios.

The run time complexity of Algorithm 2 is $O(3^{n/3})+O(n\times l)+O(s^2\times l)$, where n is the number of alerts, s is the number of alerts shared between candidate attack scenarios, and l is the number of candidate attack scenarios in ACG.

4.3 Attack Causality Analysis

The semantic clustering only groups alerts that belong to the same attack scenario into one cluster. Likewise, the candidate attack scenarios generated from the semantic clustering do not provide any information about the sequencing of the attack or the steps the attacker executes to reach his objective. However, the main goal of the attack scenario reconstruction is to identify the sequence of steps and actions taken by the intruder to break into the system. An effective way to elicit the attack sequencing consists of analyzing the causality between the individual attacks reported in the IDS alerts.

To detect the causality between different attack instances, each attack instance is associated with both a set of prerequisites and a set of consequences. The attack prerequisites are the set of logical conditions to be satisfied for the attack to succeed while the attack consequences are the set of logical conditions that will become true when the attack succeeds. Two attacks a and b are causally related if at least one of the consequences of one of them is among the prerequisites of the second one.

Algorithm 2. Extracting Candidate Attack Scenario from ACG

```
   /* A a set of IDS alerts                                              */
   /* G a matrix represent the ACG                                       */
   /* C a set of maximum clique in ACG                                   */
   /* A' a set of alerts that belong to more than one clique             */
   /* m membership between an alert a and clique c                       */
   /* n number of alerts or vertices in ACG                              */
   /* s number of alerts in A'                                           */
   /* l number of maximum cliques in ACG                                 */
   Input: G
   Output: C
 1 begin
 2     C ← BronKerbosch(G);
 3     for i ← 1 to n do
 4         β ← 0 ;
 5         for j ← i to l do
 6             if a_i ∈ c_j then
 7                 β ← β + 1;
 8                 if β ≥ 2 then
 9                     add a_i to A';
10                     Break ;
11                 end
12             end
13         end
14     end
15     while A' ≠ ∅ do
16         max ← −1;
17         for i ← 1 to s do
18             for j ← 1 to l do
19                 m ← sum of the weights of all adjacent edges of a_i in c_j;
20                 if m ≥ max then
21                     max ← m ;
22                     sAlert ← a_i ;
23                     sClique ← c_j ;
24                 end
25             end
26         end
27         remove sAlert from A';
28         foreach clique c ∈ C do
29             if c ≠ sClique and sAlert ∈ c then
30                 remove sAlert from c;
31             end
32         end
33     end
34     return C;
35 end
```

The knowledge corresponding to the attack prerequisites and consequences is represented in the intrusion ontology by introducing attack prerequisites and attack consequences relations between the *Attack* class and the *Impact* class (see Figure 1). The attack prerequisites and consequences are defined as subclasses of the *Impact* class. For any two attack instances a and b, if there is an impact p where p is a consequence of a and a prerequisite of b, then there is a causality relationship between a and b. In other words the intruder will execute first a and then b. For instance, the success of a scanning attack that detects the presence of a vulnerable FTP server is a prerequisite for a buffer overflow attack against this FTP server. It is not possible for an intruder to execute the buffer overflow

attack before the scanning attack. Now, let A denote the set of consequences of attack a and let B denote the set of prerequisites of attack b. We define the strength of the causality relation between a and b as a value between 0 and 1 given by equation 2, where 0 indicates no causality and 1 indicates maximum causality:

$$causality(a, b) = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

The process of detecting attack causality and reconstructing the attack scenario graph can be described as a graph transformation operation. The attack causality detection algorithm converts the complete graph representing the candidate attack scenario into a directed acyclic graph representing the reconstructed attack scenario. The transformation consists of simply replacing the edges in the alerts correlation graph corresponding to the semantic relevance relations between alerts with new edges that represent the causality relations between the attacks reported by the alerts.

Algorithm 3 describes the key steps of the attack causality analysis. The algorithm takes a clique (i.e. a candidate attack scenario) as an input and generates an attack scenario graph as an output. The input clique is represented by a vector V of alerts sorted in ascending order based on their timestamps. The output of the algorithm is an attack scenario graph represented by a set of matrices denoted M. The algorithm starts by creating an empty matrix m_1 and inserts the first alert in V into m_1. Then the algorithm iterates $n-1$ times, where n is the size of V. In each iteration, the algorithm checks the causality between one alert a_i from V and every alert b in every matrix m_j in M using equation 2. If the causality measure equal zero for every alert in every matrix m_j in M, the algorithm creates a new matrix m_{j+1} and adds a_i to this matrix. If the causality measure is greater than zero then the algorithm will add a_i to the matrix that returns the maximum causality with a_i.

The ideal output of the algorithm is the case where M contains a single matrix, which means that the attack scenario graph is a connected graph. The case where M contains more than one matrix indicates that the attack scenario graph is not a connected graph, which corresponds either to a false negative, a novel attack, or some missing causality information.

5 Experimental Evaluation

To evaluate our approach, we use two different datasets widely used in the literature, namely, the DARPA 2000 dataset from MIT Lincoln Laboratory [6] and the Treasure Hunt dataset [12]. Specifically, we used the LLDDOS1.0 subset of the DARPA dataset and the DMZ partition from the Treasure Hunt dataset. We analyzed the tcpdump files of the datasets using SNORT IDS version 2.9.2.0 running on Ubuntu box. Table 3 shows a summary of the contents of the datasets after analyzing them with SNORT IDS. These include the number of alerts (including redundant alerts) generated by SNORT for each dataset, the number

Algorithm 3. Attacks Causality Analysis

```
   /* V a sorted vector of alerts that belong to one clique                    */
   /* M a set of matrices that represent the attack scenario graph             */
   /* n number of alerts in V                                                  */
   /* l number of matrices in M                                                */
   Input: V
   Output: M
 1 begin
 2     create m_1 as an empty matrix in M;
 3     add V[1] to m_1;
 4     l ← 1;
 5     for i ← 2 to n do
 6         max ← 0;
 7         for j ← 1 to l do
 8             foreach alert b ∈ m_j do
 9                 δ ← causality(a_i,b);
10                 if δ > max then
11                     max ← δ ;
12                     sMatrix ← m_j ;
13                     sAlert ← b;
14                 end
15             end
16         end
17         if max ≠ 0 then
18             add a_i to m_j at sAlert;
19         else
20             l ← l + 1;
21             create m_l as an empty matrix in M;
22             add a_i to m_l;
23         end
24     end
25     return M;
26 end
```

Table 3. Datasets Statistics

Dataset	LLDDOS1.0	Hunt-DMZ
Alerts	2170	671848
Intrusions	16	49
Sources	273	28
Targets	738	37
Duration	≈ 100 minutes	≈ 893 minutes

of unique intrusions or attacks reported by SNORT, the number of source and destination IP addresses and the duration of generated network traffic.

We used the *soundness* and the *completeness* metrics, described earlier in the Introduction, to calculate the performance of our proposed approach.

By applying our approach to the DMZ partition of the treasure hunt dataset, 6 attack scenarios were detected, five of which were attack true attack scenarios and one was a false attack scenario. The true attack scenarios detected by our approach are the following: ***Protocol Exploit***, ***Reconnaissance***,***Privilege Escalation***, and two ***Web Exploit*** attack scenarios. All of these attacks target two machines inside the DMZ, while their sources are from 2 different subnets. The attackers kept executing these attack scenarios in a brute-force manner over a period of 15 hours. The false attack scenario is ***MySQL Root Attack***. The source of that attack is one machine inside the DMZ network and the target is a host in one of the Treasure Hunt internal networks.

We found that out of the total number of alerts (i.e. 671848), there are 628956 alerts related to the five attack scenarios. The remaining 42892 alerts are either false positives or single attack attempts that are irrelevant to any of the five attack scenarios. Our approach correlates 629426 alerts, 470 of which are alerts that are incorrectly considered part of the related alerts. Table 4 summarizes the performance results obtained for the different attack scenarios for the treasure hunt dataset.

Table 4. Evaluation Results with the Treasure hunt Dataset

Scenario	**Correlated alerts**	**True alerts**	**Related alerts**	**Completeness**	**Soundness**
Web Exploit 1	503337	503337	503337	100.00%	100.00%
Web Exploit 2	101071	100758	100758	100.00%	99.69%
Protocol Exploit	1730	1701	1705	99.77%	98.32%
Reconnaissance	3097	2973	3053	97.38%	96.00%
Privilege Escalation	20191	19981	20103	99.39%	98.96%

To compare our approach to previous approaches we used the LLDDOS1.0 attack scenario from the DARPA dataset, since most of the previous approaches used that dataset for evaluation. Table 5 shows the completeness and the soundness of our approach in comparison to previous works.

Table 5. Comparison of Attack Scenario Reconstruction Approaches Using the LLDDOS1.0 Dataset

Approach	**Completeness**	**Soundness**
Ning et al	93.96%	93.96%
Liu et al	87.12%	86.27%
Al-Mamory and Zhang	86.5%	100%
Li et al	92.2%	not provided
Our Approach	100%	99.70%

As shown by Tables 5 and 4, our approach outperforms many of the previous approaches. The completeness of our approach is promising and shows that our approach can correlate alerts that belong to the same attack scenario with high detection rate. At the same time the soundness of our approach is in general better than most of the previous approaches.

6 Conclusion

We have introduced in this paper a new attack scenario reconstruction technique using semantic and causality analysis. Our approach using semantic relevance to correlate related alerts based on their semantics. Experimental evaluation of our approach yields better results compared to previous works in the area of attack scenario reconstruction. Future work will aim at improving the run time of our approach and investigate the possibility of validating IDS alerts to

effectively remove false positives and irrelevant alerts. In addition, predicting missing attack steps that result from IDS false negatives is another direction for future work. Missing attack steps can prevent or hinder the reconstruction of true attack scenario, therefore predicting missing attack steps is an essential requirement to improve the attack scenario reconstruction.

References

1. Abdoli, F., Kahani, M.: Using attacks ontology in distributed intrusion detection system. In: SCSS (1), pp. 153–158 (2007)
2. Al-Mamory, S.O., Zhang, H.L.: Scenario discovery using abstracted correlation graph. In: 2007 International Conference on Computational Intelligence and Security, pp. 702–706 (December 2007)
3. Ding, Y.-X., Wang, H.-S., Liu, Q.-W.: Intrusion scenarios detection based on data mining. In: 2008 International Conference on Machine Learning and Cybernetics, vol. 3, pp. 1293–1297 (July 2008)
4. Isaza, G.A., Castillo, A.G., Duque, N.D.: An Intrusion Detection and Prevention Model Based on Intelligent Multi-Agent Systems, Signatures and Reaction Rules Ontologies. In: Demazeau, Y., Pavón, J., Corchado, J.M., Bajo, J. (eds.) 7th International Conference on PAAMS 2009. AISC, vol. 55, pp. 237–245. Springer, Heidelberg (2009)
5. Li, W., Zhi-tang, L., Dong, L., Jie, L.: Attack scenario construction with a new sequential mining technique. In: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, SNPD 2007, July 30-August 1, vol. 1, pp. 872–877 (2007)
6. Lincoln-Laboratory-MIT. Darpa intrusion detection evaluation, `http://www.ll.mit.edu/mission/communications/ist/CST/index.html`
7. Liu, Z., Wang, C., Chen, S.: Correlating multi-step attack and constructing attack scenarios based on attack pattern modeling. In: International Conference on Information Security and Assurance, ISA 2008, pp. 214–219 (April 2008)
8. Ning, P., Cui, Y., Reeves, D.S.: Constructing attack scenarios through correlation of intrusion alerts. In: CCS 2002: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 245–254. ACM, New York (2002)
9. Rhee, S.K., Lee, J., Park, M.-W.: Semantic relevance measure between resources based on a graph structure. In: International Multiconference on Computer Science and Information Technology, IMCSIT 2008, pp. 229–236 (October 2008)
10. Ruotsalo, T., Hyvonen, E.: A Method for Determining Ontology-Based Semantic Relevance. In: Wagner, R., Revell, N., Pernul, G. (eds.) DEXA 2007. LNCS, vol. 4653, pp. 680–688. Springer, Heidelberg (2007)
11. Saad, S., Traore, I.: Method ontology for intelligent network forensics analysis. In: Eight International Conference on Privacy, Security and Trust (PST 2010), Ottawa, Canada, pp. 7–14 (August 2010)
12. UCSB. The 2002 UCSB treasure hunt dataset, `http://ictf.cs.ucsb.edu/data/treasurehunt2002/`
13. Undercoffer, J.L., Joshi, A., Finin, T., Pinkston, J.: A Target-Centric Ontology for Intrusion Detection. In: The 18th International Joint Conference on Artificial Intelligence (July 2003)

On Securely Manipulating XML Data

Houari Mahfoud and Abdessamad Imine

University of Lorraine and INRIA-LORIA
Nancy, France
{Houari.Mahfoud,Abdessamad.Imine}@loria.fr

Abstract. Over the past years several works have proposed access control models for XML data where only read-access rights over non-recursive DTDs are considered. A small number of works have studied the access rights for updates. In this paper, we present a general and expressive model for specifying access control on XML data in the presence of the update operations of W3C XQuery Update Facility. Our approach for enforcing such update specification is based on the notion of *query rewriting*. A major issue is that, in practice, query rewriting for recursive DTDs is still an open problem. We show that this limitation can be avoided using only the expressive power of the standard XPath, and we propose a linear algorithm to rewrite each update operation defined over an arbitrary DTDs (recursive or not) into a safe one in order to be evaluated only over the XML data which can be updated by the user. To our knowledge, this work is the first effort for securely updating XML in the presence of arbitrary DTDs, a rich class of update operations, and a significant fragment of XPath.

Keywords: XML Access control, XML Updating, Query Rewriting, XPath, XQuery.

1 Introduction

The XQuery Update Facility language [1] is a recommendation of W3C that provides a facility to modify some parts of an XML document and leave the rest unchanged, and this through different update operations. This includes rename, insert, replace and delete operations at the node level. The security requirement is the main problem when manipulating XML documents. An XML document may be queried and/or updated simultaneously by different users. For each class of users some rules can be defined to specify parts of the document which are accessible to the users and/or updatable by them. A bulk of work has been published in the last decade to secure the XML content, but only read-access rights has been considered over non-recursive DTDs [2–5]. Moreover, a few works have considered update rights [4, 6, 7].

In this paper, we investigate a general approach for securing XML update operations of the XQuery Update Facility language. Abstractly, for any update operation posed over an XML document, we ensure that the operation is performed only on XML nodes that can be updated by the user. Addressing such

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 293–307, 2013.

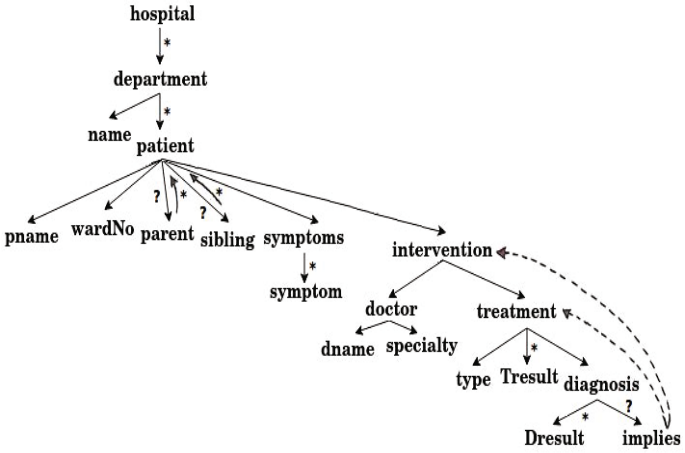


Fig. 1. Hospital DTD

concerns requires first a specification model to define update constraints and a flexible mechanism to enforce these constraints at update time.

We now discuss a motivating example for access control with updates. Consider the recursive DTD[1] depicted as a graph in Fig.1. We use '$*$' on an edge to indicate a list, '?' to indicate optional edge, while dashed edges represent disjunction. A hospital document conforming to this DTD consists of a list of departments (*dept*) defined by a *name*, and each department has a list of children representing patients currently residing in the hospital. For each *patient*, the hospital maintains her name (*pname*), ward number (*wardNo*), family medical history by means of the recursively defined *parent* and *sibling*, as well as list of *symptoms*. The hospitalization is marked by the *intervention* of one or many doctors depending on their specialty and the patient care requirement. For each intervention, the hospital also maintains the information of the responsible *doctor* (defined with name (*dname*) and *specialty*) and the *treatment* applied. A treatment is described by its *type*, a list of result (*Tresult*), and it is followed by a *diagnosis* phase. According to the results of the diagnosis (*Dresult*), the doctor may decide to do another treatment. However, if the required treatment is outside his area of expertise, then the current doctor would solicit the intervention of another doctor, specialist, or expert.

An instance of the hospital DTD is given in Fig. 2. Due to space limitation, this instance is split into two parts. Figure2 (a) represents a simple hospital document with *Cardiology* department, *Critical care* department, as well as some patients information of these departments[2]. Figure2 (b) depicts the three interventions done for $patient_1$: $intervention_1$, $intervention_2$, and $intervention_3$.

[1] A DTD is recursive if and only if at least one of its elements is defined (directly or indirectly) in terms of itself.

[2] We use the notation X_i to distinguish between different instances of element type X, like $patient_1$.

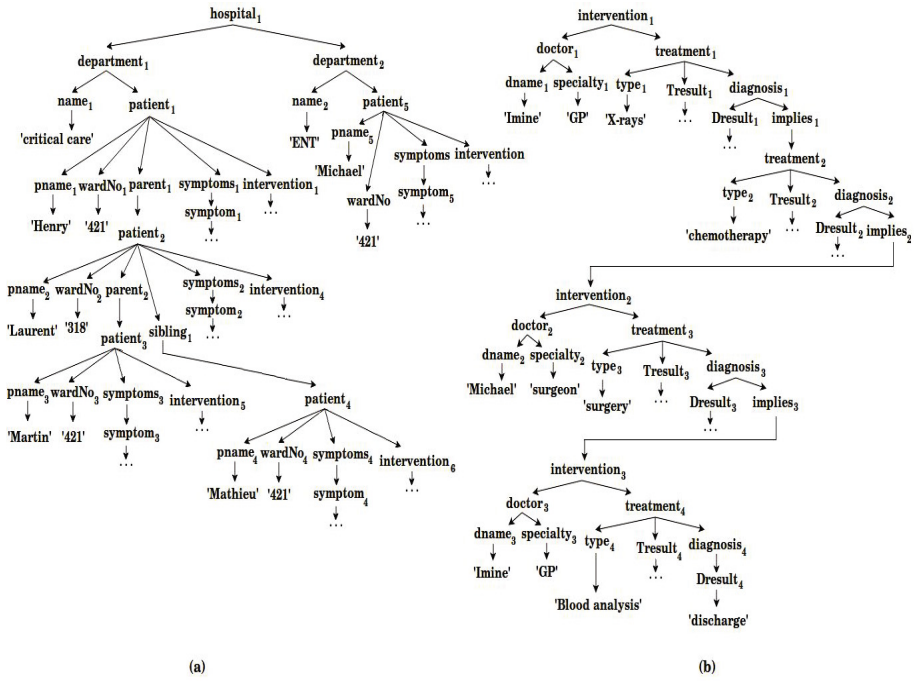


Fig. 2. Hospital Data: (a) patients information, (b) interventions done for $patient_1$

Example 1. (*Update Policy for doctors*) Suppose that the hospital wants to impose a security policy that authorizes each doctor to update only the information of treatments that she has done. For instance, the doctor *Imine* could update the data of $treatment_1$, $treatment_2$, and $treatment_4$ (like insert new *Dresult* sub-tree into the node $diagnosis_4$) but not $treatment_3$. We show in the following that this update policy, even simple, cannot be enforced by using some existing update specification languages. □

Problem 1. (*Expressiveness of Update Specification Languages*) In some case of recursive DTDs, the existing update access control models are unable to specify some update policies. In the model proposed by Damiani et al. [4], the update policy is defined by annotating the XML schema by security attributes. For instance, adding attribute `@insert=[`*test='Blood Analysis'*`]` into element type *treatment* of the hospital DTD specifies that new sub-tree can be inserted to *treatment* nodes having '*Blood Analysis*' as *type*. However, only local annotations can be defined (i.e. the update constraint concerns only the node and not its descendants) which makes the proposed model restricted for non-recursive schema/DTD. For instance, the updates of doctor *Imine* cannot be discarded for the node $treatment_3$ as imposed by the update policy defined above. Specifically, adding attribute `@insert=[ancestor::`*intervention*`[`*doctor/dname='Imine'*`]]` into *treatment*

element type makes all treatment nodes updatable by *Imine*. Since the hospital DTD is recursive, this update policy cannot be specified by the model proposed in [4]. To specify the imposed update policy, a plausible solution may be done by using the *transitive closure operator* '*'. In this case, the adequate update constraint would be defined by adding the following attribute into *treatment* element type:

`@insert=[(parent::`*implies*`/parent::`*diagnosis*`/parent::`*treatment*`)*/`
`parent::`*investigation*`[`*doctor*`/`*dname*`=`$DNAME`]]`

Where $DNAME is treated as a constant parameter; i.e., when a concrete value, e.g., *Imine*, is substituted for $DNAME, the previous annotation defines the update right for doctor *Imine*. However, the *transitive closure operator* cannot be expressed in the standard XPath as outlined in [8].

Due to space constraints, we do not discuss about the limitation of the update access control model (called `XACU`) proposed in [6]. For more details, the reader is referred to our extended version available online [3].

To the best of our knowledge, no model exists for specifying update policies over recursive DTDs.

Problem 2. (*Query Rewriting Limitation*) For each update operation, an XPath expression is defined to specify the XML data at which the update is applied. To enforce an update policy, the *query rewriting* principle can be applied where each update operation (i.e., its XPath expression) is rewritten according to the update constraints into a safe one in order to be performed only over parts of the XML data that can be updated by the user who submitted the operation. However, this rewriting step is already challenging for a small class of XPath. Consider the *downward* fragment of XPath which supports *child* and *descendant-or-self* axes, union and complex predicates. In case of recursive DTDs, it was shown that an XPath expression defined in this fragment cannot be rewritten safely. More specifically, a safe rewriting of the XPath expression of an update operation can stand for an infinite set of paths which cannot be expressed in the downward fragment of XPath (even by using the upward-axes: *parent*, *ancestor*, and *ancestor-or-self*).

To overcome this rewriting limitation, one can use the '*Regular XPath*' language [9], which includes the *transitive closure operator* and allows to express recursive paths. However, it remains a theoretical achievement since no tool exists to evaluate Regular XPath queries. Thus, no practical solution exists for enforcing update policies in the presence of recursive DTDs.

Our Contributions. Our first contribution is an expressive model for specifying XML update policies, based on the primitives of the XQuery Update Facility, and over arbitrary DTDs (recursive or not). Given a DTD D, we annotate element types of D with different update rights to specify restrictions on updating some parts of XML documents that conform to D. Each update right concerns one update operation (e.g., deny insertion of new nodes of type $Tresult$ under

[3] `http://hal.inria.fr/hal-00664975`

treatment nodes). Our model supports *inheritance* and *overriding* of update privileges and overcomes expressivity limitations of existing models (see **Problem 1**). Our approach for enforcing such update policies is based on the notion of *query rewriting*. However, to overcome the rewriting limitation presented above as **Problem 2**, we investigate the extension of the downward fragment of XPath using upward-axes and position predicate. Based on this extension, our second contribution is a linear algorithm that rewrites any update operation defined in the downward fragment of XPath into another one defined in the extended fragment to be safely performed over the XML data. To our knowledge, this yields the first model for specifying and enforcing update policies using the XQuery update operations and in the presence of arbitrary DTDs.

Related Work. During the last years, several works have proposed access control models to secure XML content, but only read-access has been considered over non-recursive DTDs [2–4]. There has been a few amount of work on securing XML data by considering the update rights. Damiani et al. [4] propose an XML access control model for update operations of the XUpdate language. They annotate the XML schema with the read and update privileges, and then the annotated schema is translated into two automatons defining read and update policies respectively, which are used to rewrite any access query (resp. update operation) over the XML document to be safe. However, the update policy is expressed only with local annotations which is not sufficient to specify some update rights (see *Problem 1*). Additionally, the automaton processing cannot be successful when rewriting access queries (resp. update operations) defined over recursive schema (i.e., recursive DTD). Fundulaki et al. [6] propose an XML update access control model, called `XACU`, for the XQuery update operations. A set of XPath-based rules is used to specify, for each update operation, the XML nodes that can be updated by the user using this operation. In the presence of non-recursive DTD only, the `XACU` rules can be translated into annotations over element types of the DTD to present an annotation-based model called $\texttt{XACU}^{annot}$.

The view-based access control for XML data has received an increased attention [2,5,10]. However, a major issue arises in the case of recursive security views when XPath query rewriting becomes not possible. To overcome this problem, some authors [10, 11] propose rewriting approaches based on the non-standard language, "Regular XPath" [9], which is more expressive than XPath and makes rewriting possible under recursion. However, no system exists for evaluating regular XPath queries in order to demonstrate the practicality of the proposed approaches. Thus, the need of a rewriting system of XPath queries (resp. update operations) over recursion remains an open issue.

Plan of the Paper. The paper is organized as follows. Section 2 reviews some basic notions tackled throughout the paper. We describe in Section 3 our specification model of update. Our approach for securing update operations is detailed in Section 4. Finally, we conclude this paper in Section 5.

2 Background

This section briefly reviews some basic notions tackled throughout the paper.

DTDs. Without loss of generality, we represent a DTD D by (*Ele*, *Rg*, *root*), where *Ele* is a finite set of *element types*; *root* is a distinguished type in *Ele* called the *root type*; *Rg* is a function defining element types such that for any A in *Ele*, $Rg(A)$ is a regular expression α defined as follows:

$$\alpha \texttt{ := str} \mid \epsilon \mid B \mid \alpha\text{'},\text{'}\alpha \mid \alpha\text{'}|\text{'}\alpha \mid \alpha^* \mid \alpha+ \mid \alpha?$$

where `str` denotes the text type `PCDATA`, ϵ is the empty word, B is an element type in *Ele*, α','α denotes concatenation, and α'|'α denotes disjunction. We refer to $A \rightarrow Rg(A)$ as the *production* of A. For each element type B occurring in $Rg(A)$, we refer to B as a *sub-element type* (or *child type*) of A and to A as a *super-element type* (or *parent type*) of B. The sub-elements structure can be specified using the operators '*' (set with zero or more elements), '+' (set with one or more elements), and '?' (optional set of elements). A DTD D is *recursive* if some element type A is defined in terms of itself directly or indirectly.

As depicted in Fig. 1, our DTD graph representation is specified with solid edges (which represent conjunction), dashed edges (which represent disjunction). These edges can be labeled with one of the operators '*', '+', or '?'. This simple graph representation suffices to depict our hospital DTD. However, for a complete representation of DTDs, a special *DTD graph* structure can be used, along the same lines as [3].

XML Trees. We model an XML document with an unranked ordered finite node-labeled tree. Let Σ be a finite set of node labels, an XML document T over Σ is a structure defined as [9]: T=$(N, R_{\downarrow}, R_{\rightarrow}, L)$, where $(N, R_{\downarrow})$ is a finite rooted tree with child relation $R_{\downarrow} \subseteq N \times N$, $R_{\rightarrow} \subseteq N \times N$ is a successor relation on (ordered) siblings, and $L : N \rightarrow \Sigma$ is a function assigning to every node its label. We use the term *XML Tree* for this type of structures.

An XML tree $T = (N, R_{\downarrow}, R_{\rightarrow}, L)$ conforms to a DTD $D = (Ele, Rg, r)$ if the following conditions hold: (i) the root of T is the unique node labeled with r; (ii) each node in T is labeled either with an *Ele* type A, called an A *element*, or with `str`, called a *text node*; (ii) for each A element with k ordered children $n_1, ..., n_k$, the word $L(n_1), ..., L(n_k)$ belongs to the regular language defined by $Rg(A)$; (iv) each text node carries a string value (`PCDATA`) and is the leaf of the tree. We call T an instance of D if T conforms to D.

XPath Queries. We consider a small class of XPath [12] queries, referred to as $\mathcal{X}$ and defined as follows:

$$\begin{aligned} p &:= \alpha\texttt{::}ntst \mid p\texttt{[}q\texttt{]} \mid p\texttt{/}p \mid p \cup p \\ q &:= p \mid p\texttt{='}c\texttt{'} \mid q \wedge q \mid q \vee q \mid \neg\texttt{(}q\texttt{)} \\ \alpha &:= \varepsilon \mid \downarrow \mid \downarrow^+ \mid \downarrow^* \end{aligned}$$

where p denotes an XPath query and it is the start of the production, *ntst* is a node test that can be an element type, $*$ (that matches all types), or function *text()* (that tests whether a node is a text node), c is a string constant, and $\cup$, $\wedge$, $\vee$, $\neg$ denote *union*, *conjunction*, *disjunction*, and *negation* respectively; α stands for XPath axis relations and can be one of ε, $\downarrow$, $\downarrow^+$, or $\downarrow^*$ which denote *self*, *child*, *descendant*, and *descendant-or-self* axis respectively. Finally the expression q is called a *qualifier* or *predicate*. The result of the evaluation of an $\mathcal{X}$ query p at a *context node* n of an XML Tree T, is the set of nodes reachable via p from n, denoted by $n[\![p]\!]$. We denote by $n \vDash q$ a qualifier q that is valid at a node n.

Authors of [10] have shown that in the case of recursive security views, the fragment $\mathcal{X}$ (called *downward* fragment) is not *closed* under query rewriting. This means that is not always possible to rewrite XPath queries on views to be safely evaluated on the source. Consequently, it is also the problem of update operations rewriting since fragment $\mathcal{X}$ is the core of XQuery, XSLT and XML Schema. Our solution to make possible the update operations rewriting is based on the following extension:

$$
\begin{array}{l}
p \;:=\; \alpha::ntst \mid p[q] \mid p/p \mid p \cup p \mid p[n] \\
q \;:=\; p \mid p='c' \mid q \wedge q \mid q \vee q \mid \neg\,(q) \\
\alpha \;:=\; \varepsilon \mid \downarrow \mid \downarrow^+ \mid \downarrow^* \mid \uparrow \mid \uparrow^+ \mid \uparrow^*
\end{array}
$$

we enrich $\mathcal{X}$ by the *position* predicate and the upward-axes presented by *parent* axis ($\uparrow$), *ancestor* axis ($\uparrow^+$), and *ancestor-or-self* axis ($\uparrow^*$). The position predicate, defined with $[n](n \in \mathrm{N})$, is used to return the n^{th} node from an ordered set of nodes. For instance, the query $\downarrow::*[2]$ at a node n of an ordered returns its second child node. We denote this extended fragment with $\mathcal{X}^{\Uparrow}_{[n]}$.

In our case, fragment $\mathcal{X}$ is used only to formulate update operations and to define our security policies. While we will explain later how the fragment $\mathcal{X}^{\Uparrow}_{[n]}$ defined above can be used to avoid the rewriting limitation.

XML Update Operations. We review some update operations of the W3C XQuery Update Facility recommendation [1]. We study the use of the following operations: *insert*, *delete*, and *replace*. For each update operation, an XPath *target* expression is used to specify the set of XML node(s) in which the update is applied. In a *delete* operation, *target* specifies the XML nodes to be deleted (denoted *target-nodes*). For *insert*, and *replace* operations, *target* must specify a single node (denoted *target-node*); otherwise a dynamic error is raised. Moreover, the latter operations require a second argument *source* representing a sequence of XML nodes. The order defined between the nodes of *source* must be preserved during the insertion and replacement. In the following, names in brackets are abbreviations of the different operations.

Insert. We distinguish different types of insert operation depending on the position of the insertion:

• **insert** *source* **as first/last into** *target* [*insertAsFirst*/*insertAsLast*]: Here *target-node* must evaluate to a single element node; otherwise a dynamic

error is raised. This operation inserts the nodes in *source* as first/last children of *target-node* respectively.

• **insert** *source* **before/after** *target* [*insertBefore*/*insertAfter*]: Inserts the nodes in *source* as preceding/following sibling nodes of *target-node* respectively. In this case, *target-node* must have a parent node; otherwise a dynamic error is raised.

• **insert** *source* **into** *target* [*insertInto*]: Inserts the nodes in *source* as children of the single element node *target-node* (otherwise a dynamic error is raised). Note that the positions of the inserted nodes among the children of *target-node* are implementation-dependent [4]. Thus, the effect of executing an *insertInto* operation on *target-node* can be that of *insertAsFirst*/*insertAsLast* executed on *target-node*, or that of *insertBefore*/*insertAfter* executed at children of *target-node*.

Delete. The operation "**delete** *target*" [*delete*] deletes all *target-nodes* along with their descendant nodes.

Replace. The operation "**replace** *target* **with** *source*" [*replace*] replaces *target-node* with the nodes in *source*. Here *target-node* must have a parent node; otherwise a dynamic error is raised. If *target-node* is an element or text node, then *source* must be a sequence of elements or text nodes respectively. The *target-node* is deleted along with its descendants and replaced by the nodes in *source* together with their descendants.

3 Update Access Control Model

This section describes our access control model for XML update.

3.1 Update Specifications

We follow the idea of *security annotations* presented in [2] and the *update access types* notion introduced in [13] to define a language for specifying expressive and fine-grained XML update policies in the presence of DTDs. An *update specification* S_{up} expressed in the language is a simple extension of the document DTD D associating element types with *update annotations* (XPath qualifiers), which specify for any XML tree T conforms to D, the parts of T that can be updated by the user through a specific update operation.

Definition 1. *Given a document DTD D, an* update type (ut) *defined over D is of the form* *insertInto*$[B_i]$, *insertAsFirst*$[B_i]$, *insertAsLast*$[B_i]$, *insertBefore*$[B_i,B_j]$, *insertAfter*$[B_i,B_j]$, *delete*$[B_i]$, *and* *replace*$[B_i,B_j]$, *where* B_i *and* B_j *are element types of D.* □

[4] For instance, in the DataDirect XQuery implementation, available at `http://www.cs.washington.edu/research/xmldatasets/`, *insertInto* operation has the same effect as *insertAsLast*.

Table 1. Semantics of the update annotations Y, N, and $[Q]$

Annotation	Semantic
$ann_{up}(A, \texttt{insertInto}[B_i]) = Y\|N\|[Q]$	for a node n of type A, one can (Y)/cannot (N)/can if $n \vDash Q$, insert nodes of type B_i in an arbitrary position children of n.
$ann_{up}(A, \texttt{insertAsFirst}[B_i]) = Y\|N\|[Q]$	for a node n of type A, one can (Y)/cannot (N)/can if $n \vDash Q$, insert nodes of type B_i as first children of n.
$ann_{up}(A, \texttt{insertBefore}[B_i, B_j]) = Y\|N\|[Q]$	for a node n of type A, one can (Y)/cannot (N)/can if $n \vDash Q$, insert nodes of type B_j as preceding sibling nodes of any child node of n whose type is B_i.
$ann_{up}(A, \texttt{delete}[B_i]) = Y\|N\|[Q]$	for a node n of type A, one can (Y)/cannot (N)/can if $n \vDash Q$, delete children of n whose type is B_i.
$ann_{up}(A, \texttt{replace}[B_i, B_j]) = Y\|N\|[Q]$	for a node n of type A, one can (Y)/cannot (N)/can if $n \vDash Q$, replace children of n of type B_i by some nodes of type B_j.

Intuitively, each update type *ut* represents an update operation that is restricted to be applied only for specific element types. For example, the update type $\texttt{replace}[B_i,B_j]$ represents the update operations "**replace** *target* **with** *source*" where *target-node* is of type B_i and nodes in *source* are of type B_j.

Based on this notion of update type, we define our *update specifications* as follows:

Definition 2. *An* update specification S_{up} *is a pair* (D, ann_{up}) *where* D *is a DTD and* ann_{up} *is a partial mapping such that, for each element type* A *in* D *and each update type ut defined over element types of* D, $ann_{up}(A, ut)$, *if explicitly defined, is an annotation of the form:*

$$ann_{up}(A, ut) ::= Y \mid N \mid [Q] \mid N_h \mid [Q]_h$$

where Q *is a qualifier in our XPath fragment* $\mathcal{X}$. □

An update specification S_{up} is an extension of a DTD D associating update annotations with element types of D. In a nutshell, a value of Y, N, or $[Q]$ for $ann_{up}(A, ut)$ indicates that, for A elements in an instantiation of D, the user is *authorized*, *unauthorized*, or *conditionally authorized* respectively, to perform update operations of type *ut* at A (case of *insertInto*, *insertAsFirst*, or *insertAsLast* operations) or at children of A (case of the remaining operations). Table 1 presents more specifically the semantics of the update annotations Y, N, and $[Q]$[5].

Our model supports *inheritance* and *overriding* of update annotations. If $ann_{up}(A,ut)$ is not explicitly defined, then an A element *inherits* from its parent node the update authorization that concerns the same update type *ut*. On the other hand, if $ann_{up}(A,ut)$ is explicitly defined it may *override* the inherited authorization of A that concerns the same update type *ut*. All update operations are not permitted by default.

[5] The semantics of annotations with the update types $ann_{up}(A, \texttt{insertAslast}[B_i])$ and $ann_{up}(A, \texttt{insertAfter}[B_i, B_j])$ are defined in a similar way as $ann_{up}(A, \texttt{insertAsFirst}[B_i])$ and $ann_{up}(A, \texttt{insertBefore}[B_i, B_j])$ respectively.

Table 2. Semantics of downward-closed annotations (*ut* can be any update type)

Downward-closed Annotation	**Semantic**
$ann_{up}(A,ut) = N_h$	Same principle as $ann_{up}(A,ut) = N$ of Table 1. Moreover, for a node n of type A, all annotations of type ut defined over descendant types of A are discarded regardless their truth values.
$ann_{up}(A,ut) = [Q]_h$	Same principle as $ann_{up}(A,ut) = [Q]$ of Table 1. Moreover, for a node n of type A, if $n \not\models Q$ then all annotations of type ut defined over descendant types of A are discarded regardless their truth values.

Finally, the semantics of the specification values N_h and $[Q]_h$ are given in Table 2. The annotation $ann_{up}(A,ut)=N_h$ indicates that, for a node n of type A, update operations of type ut cannot be performed at any node of the subtree rooted at n, and no overriding of this authorization value is permitted for descendants of n. For instance, if n has a descendant node n' whose type is A', then an update operation with the same type ut cannot be performed at/under n' even though the annotation $ann_{up}(A',ut)=Y$ is explicitly defined (resp. $ann_{up}(A',ut)=[Q']$ with $n' \models Q'$). As for the annotation $ann_{up}(A,ut)=[Q]_h$, qualifier Q must be valid at A elements, otherwise no annotation with update type ut can override the *false* evaluation of Q. For instance, let n and n' be two nodes of type A and A' respectively, and let n' be a descendant node of n. The annotation $ann_{up}(A',ut)=[Q']$ indicates that an update operation of type ut can be performed at (children of) n' iff: $n' \models Q'$. Moreover, if the annotation $ann_{up}(A,ut)=[Q]_h$ is explicitly defined then the annotation $ann_{up}(A',ut)=[Q']$ takes effect at descendant node n' of n only if $n \models Q$. This means that an update operation of type ut can be performed at (children of) n' iff: $(n \models Q \wedge n' \models Q')$. We call annotation with value N_h or $[Q]_h$ as *downward-closed* annotation.

Example 2. Suppose that each nurse is attached to only one department and only one ward within this department (denoted $NURSEDEPT and $NURSEWARDNO resp.). Now, the hospital wants to impose an update policy that allows a nurse to update data of only patients having the same ward number as her (*Rule1*) and which are being treated at her department (*Rule2*). Moreover, all sibling data cannot be updated (*Rule3*). This policy can be specified by the following update annotations (*ut* denotes a general update type):

R_1: $ann_{up}(department,ut)=[\downarrow::name=\$\text{NURSEDEPT}]_h$
R_2: $ann_{up}(patient,ut)=[\downarrow::wardNo=\$\text{NURSEWARDNO}]$
R_3: $ann_{up}(sibling,ut)=N_h$

Consider the case of the nurse having the ward number *421* and working at *Critical care* department, and let *ut* be **`delete`**[*symptom*]. This nurse can delete all symptoms of Fig. 2 except: $symptom_2$ (since $patient_2$ has ward number *318*), $symptom_4$ (representing part of sibling data), and $symptom_5$ (although $patient_5$

has ward number *421*, he is attached to *ENT* department). Notice that the annotations R_1 and R_3 must be defined as *downward-closed* to enforce the imposed policy, otherwise annotation R_2 overrides at nodes $patient_4$ and $patient_5$ the negative authorizations inherited respectively from the nodes $sibling_1$ and $department_2$, which violates the imposed policy and makes possible the deletion of the nodes $symptom_4$ and $symptom_5$. □

3.2 Rewriting Problem

As will be seen shortly, in the case of recursive DTDs, update operations rewriting is already challenging for the small fragment $\mathcal{X}$ of XPath. Recall the update policy defined in Example 1. In our case, this policy can be specified by defining only the following update annotation:

$ann_{up}(intervention, ut) = [\downarrow::doctor/\downarrow::dname=\$\text{DNAME}]$

Where $DNAME is a constant parameter representing doctor's name, and *ut* can be any update type relevant to the update rules of Example 1. Now, let $DNAME be *Imine* and *ut* be ***delete***$[Tresult]$. The update operation ***delete*** $\downarrow^+::treatment[\downarrow::type='chemotherapy']/\downarrow::Tresult$ cannot be rewritten in $\mathcal{X}$ to be safe. Indeed, the *Tresult* nodes that doctor *Imine* is authorized to delete can be represented by an infinite set of paths. This latter can be captured by rewriting the previous update into the following one: ***delete*** $\downarrow^+::intervention[\downarrow::doctor/\downarrow::dname=\$\text{DNAME}]/(\downarrow::treatment/\downarrow::diagn\text{-}osis/\downarrow::implies)^*/\downarrow::treatment[\downarrow::type='chemotherapy']/\downarrow::Tresult$, defined in Regular XPath and which, when evaluated on the XML tree of Fig. 2, delete only the node $Tresult_2$. However, the Kleene star cannot be expressed in XPath [9].

We explain in the next section how the extended fragment $\mathcal{X}^{\Uparrow}_{[n]}$, defined in Section 2, can be used to overcome this rewriting limitation of update operations.

4 Securely Updating XML

In this section we focus only on update rights and we assume that every node is read-accessible by all users. Given an update specification $S_{up}=(D, ann_{up})$, we discuss the enforcement of such update constraints where each update operation posed over an instance T of D must be performed only at the nodes of T that can be updated by the user w.r.t. S_{up}. We assume that the XML tree T remains valid after the update operation is performed, otherwise the update is rejected. In the following, we denote by S_{ut} the set of annotations defined in S_{up} with the update type *ut* and by $|S_{ut}|$ the size of this set. Moreover, for an annotation function *ann*, we denote by $\{ann\}$ the set of all annotations defined with *ann*, and by $|ann|$ the size of this set.

4.1 Updatability

We say that a node n is *updatable* w.r.t. update type *ut* if the user is granted to perform update operations of type *ut* either at node n (case of *insert* operations)

or over children nodes of n (case of *delete* and *replace* operations). For instance, if a node n is updatable w.r.t. $\texttt{insertInto}[B]$, then some nodes of type B can be inserted as children of n. Moreover, B_i children of n can be replaced with nodes of type B_j iff n is updatable w.r.t. $\texttt{replace}[B_i,B_j]$.

Definition 3. *Let S_{up}=(D, ann_{up}) be an update specification and ut be an update type. A node n in an instantiation of D is* updatable *w.r.t. ut if the following conditions hold:*

i) The node n is concerned by a valid annotation[6] *with type ut; or, no annotation of type ut is defined over element type of n and there is an ancestor node n' of n such that: n' is the first ancestor node of n concerned by an annotation of type ut, and this annotation is valid at n' (called the* inherited annotation*).*

ii) There is no ancestor node of n concerned by an invalid downward-closed annotation of type ut. □

Given an update specification S_{up}=(D, ann_{up}), we define two predicates $\mathcal{U}^1_{ut}$ and $\mathcal{U}^2_{ut}$ (expressed in fragment $\mathcal{X}^{\Uparrow}_{[n]}$) to satisfy the conditions (i) and (ii) of Definition 3 with respect to an update type ut:

$$\mathcal{U}^1_{ut} := \uparrow^*::*[\vee_{(ann_{up}(A,ut)=Y|N|[Q]|N_h|[Q]_h)\in S_{ut}}\ \varepsilon::A][1]$$
$$[\vee_{(ann_{up}(A,ut)=Y)\in S_{ut}}\ \varepsilon::A\ \vee_{(ann_{up}(A,ut)=[Q]|[Q]_h)\in S_{ut}}\ \varepsilon::A[Q]]$$

$$\mathcal{U}^2_{ut} := \wedge_{(ann_{up}(A,ut)=N_h)\in S_{ut}}\ \text{not}\ (\uparrow^+::A)$$
$$\wedge_{(ann_{up}(A,ut)=[Q]_h)\in S_{ut}}\ \text{not}\ (\uparrow^+::A[not(Q)])$$

The predicate $\mathcal{U}^1_{ut}$ has the form $\uparrow^*::*[qual_1][1][qual_2]$. Applying $\uparrow^*::*[qual_1]$ on a node n returns an ordered set $\mathcal{S}$ of nodes (node n and/or some of its ancestor nodes) such that for each one an annotation of type ut is defined over its element type. The predicate $\mathcal{S}[1]$ returns either node n, if an annotation of type ut is defined over its element type; or the first ancestor node of n concerned by an annotation of type ut. Thus, to satisfy condition (i) of Definition 3, it amounts to check that the node returned by $\mathcal{S}[1]$ is concerned by a valid annotation of type ut; checked by the predicate $\mathcal{S}[1][qual_2]$ (i.e., $n \vDash \mathcal{U}^1_{ut}$). The second predicate is used to check that all downward-closed annotations of type ut defined over ancestor nodes of n are valid (i.e., $n \vDash \mathcal{U}^2_{ut}$).

Definition 4. *Let S_{up}=(D, ann_{up}), ut, and T be an update specification, an update type and an instance of DTD D respectively. We define the* updatability predicate $\mathcal{U}_{ut}$ *which refers to an $\mathcal{X}^{\Uparrow}_{[n]}$ qualifier such that, a node n on T is* updatable *w.r.t. ut iff $n \vDash \mathcal{U}_{ut}$, where $\mathcal{U}_{ut} := \mathcal{U}^1_{ut} \wedge \mathcal{U}^2_{ut}$.* □

For example, the XPath expression $\downarrow^+::*[\mathcal{U}_{ut}]$ stands for all nodes which are updatable w.r.t. ut. As a special case, if $S_{ut} = \phi$ then $\mathcal{U}_{ut} = false$.

[6] Note that an annotation $ann_{up}(A, ut)$=$value$ is *valid* at a node n if this latter is of type A and either $value$=Y; or, $value$=$[Q]/[Q]_h$ and $n \vDash Q$.

Property 1. For an update specification S_{up}=(D, ann_{up}) and an update type ut, the updatability predicate $\mathcal{U}_{ut}$ can be constructed in at most $O(|ann_{up}|)$ time. Moreover, $|\mathcal{U}_{ut}|$=$O(|ann_{up}|)$. □

Example 3. Consider the three update annotations R_1, R_2, and R_3 defined in Example 2, and let the update type ut be **delete**[symptom]. According to these annotations, the predicate $\mathcal{U}_{ut} := \mathcal{U}^1_{ut} \wedge \mathcal{U}^2_{ut}$ is defined with:

$$\mathcal{U}^1_{\textbf{delete}[symptom]} := \uparrow^*::*[\varepsilon::department \vee \varepsilon::patient \vee \varepsilon::sibling][1] \\ [\varepsilon::department[\downarrow::name=\$\text{NURSEDEPT}] \\ \vee \varepsilon::patient[\downarrow::wardNo=\$\text{NURSEWARDNO}]]$$

$$\mathcal{U}^2_{\textbf{delete}[symptom]} := \text{not } (\uparrow^+::department[\text{not } (\downarrow::name=\$\text{NURSEDEPT})]) \wedge \\ \text{not } (\uparrow^+::sibling)$$

Consider the case of the nurse having the ward number *421* and working at *Critical care* department. The predicate $\uparrow^*::*[\varepsilon::department \vee \varepsilon::patient \vee \varepsilon::sibling]$ over the node $patient_3$ of Fig. 2 returns the ordered set $\mathcal{S}$={$patient_3$, $patient_2$, $patient_1$, $department_1$} of nodes (each one is concerned by an annotation of type **delete**[symptom]); $\mathcal{S}[1]$ returns $patient_3$ and the predicate [$\varepsilon::department[\downarrow::name$='*Critical care*'] $\vee$ $\varepsilon::patient[\downarrow::wardNo$='*421*']] is valid at node $patient_3$ (i.e. $patient_3 \vDash \mathcal{U}^1_{\textbf{delete}[symptom]}$). Also, we can see that $patient_3 \vDash \mathcal{U}^2_{\textbf{delete}[symptom]}$. Consequently, the node $patient_3$ is updatable w.r.t. **delete**[*symptom*] (i.e., $patient_3 \vDash \mathcal{U}_{\textbf{delete}[symptom]}$). This means that the nurse is granted to delete *symptom* elements of $patient_3$ (e.g. node $symptom_3$). However, for node $patient_5$ we can check that the predicate $\mathcal{U}^1_{\textbf{delete}[symptom]}$ is valid, while it is no longer the case for the predicate $\mathcal{U}^2_{\textbf{delete}[symptom]}$ ($patient_5$ has an ancestor node $department_2$ with $name \neq$ '*Critical care*'). Thus, the nurse is not allowed to delete the node $symptom_5$. □

4.2 Rewriting of Update Operations

Finally, we detail here our approach for enforcing update policies based on the notion of "*query rewriting*". Given an update specification S_{up}=(D, ann_{up}). For any update operation with *target* defined in the XPath fragment $\mathcal{X}$, we translate this operation into a safe one by rewriting its *target* expression into another one *target*′ defined in the XPath fragment $\mathcal{X}^{\Uparrow}_{[n]}$, such that evaluating *target*′ over any instance T of D returns only nodes that can be updated by the user w.r.t. S_{up}. We describe in the following the rewriting of each kind of update operation considered in this paper, where DTD $D = (Ele, Rg, root)$ and *source* is a sequence of nodes of type B_j.

• " **delete** *target* ": For any node n of type B_i referred to by *target*, parent node n' of n must be updatable w.r.t. **delete**[B_i] (i.e., $n' \vDash \mathcal{U}_{\textbf{delete}[B_i]}$). To satisfy this, we rewrite *target* expressions of **delete** operations into: $target[\vee_{B_i \in Ele} \varepsilon::B_i[\uparrow::*[\mathcal{U}_{\textbf{delete}[B_i]}]]]$.

• " **replace** *target* **with** *source* ": A node n of type B_i referred to by *target* can be replaced with the nodes in *source* iff its parent node n' is updatable w.r.t.

$\texttt{replace}[B_i,B_j]$ (i.e., $n' \vDash \mathcal{U}_{\texttt{replace}[B_i,B_j]}$). Thus, *target* expressions of `replace` operations can be rewritten into:
$target[\vee_{B_i \in Ele}\ \varepsilon::B_i[\uparrow::*[\mathcal{U}_{\texttt{replace}[B_i,B_j]}]]]$.

• " **insert** *source* **before/after** *target* ": For any node n referred to by *target*, the user can insert nodes in *source* as preceding sibling nodes of n iff its parent node n' is updatable w.r.t. $\texttt{insertBefore}[B_i,B_j]$ (i.e., $n' \vDash \mathcal{U}_{\texttt{insertBefore}[B_i,B_j]}$). To satisfy this, *target* expressions of `insertBefore` operations can be rewritten into: $target[\vee_{B_i \in Ele}\ \varepsilon::B_i[\uparrow::*[\mathcal{U}_{\texttt{insertBefore}[B_i,B_j]}]]]$. The same principle is applied for `insertAfter` operations.

• " **insert** *source* **into** *target* ": For any node n referred to by *target*, the user can insert nodes in *source* as children of n (in an implementation-dependent position), provided that he holds the $\texttt{insertInto}[B_j]$ right on this node (i.e., $n \vDash \mathcal{U}_{\texttt{insertInto}[B_j]}$). To check this, *target* expressions of `insertInto` operations can be simply rewritten into: $target[\mathcal{U}_{\texttt{insertInto}[B_j]}]$. The same principle is applied for `insertAsFirst` and `insertAsLast` operations.

Theorem 1. *For any update specification* $S_{up}=(D, ann_{up})$ *and any update operation with* target *expression defined in* $\mathcal{X}$*, there exists an algorithm "*`Updates Rewrite`*" that translates* target *into a safe one* target' *(defined in* $\mathcal{X}^{\Uparrow}_{[n]}$*) in at most* $O(|D| + |ann_{up}|)$ *time. Moreover,* $|target'|=O(|target| + |ann_{up}|)$. □

5 Conclusion

We have proposed a general model for specifying XML update policies based on the primitives of the XQuery Update Facility. To enforce such policies, we have introduced a rewriting approach to securely updating XML over arbitrary DTDs and for a significant fragment of XPath. To our knowledge, this paper presents the first work for securely updating XML data over general DTDs.

References

1. Robie, J., Chamberlin, D., Dyck, M., Florescu, D., Melton, J., Siméon, J.: Xquery update facility 1.0 (March 2011), `http://www.w3.org/TR/xquery-update-10/`
2. Fan, W., Chan, C.Y., Garofalakis, M.N.: Secure XML querying with security views. In: ACM SIGMOD (2004)
3. Kuper, G.M., Massacci, F., Rassadko, N.: Generalized XML security views. Int. J. Inf. Sec. 8(3), 173–203 (2009)
4. Damiani, E., Fansi, M., Gabillon, A., Marrara, S.: A general approach to securely querying XML. Computer Standards & Interfaces 30(6), 379–389 (2008)
5. Rassadko, N.: Policy Classes and Query Rewriting Algorithm for XML Security Views. In: Damiani, E., Liu, P. (eds.) Data and Applications Security 2006. LNCS, vol. 4127, pp. 104–118. Springer, Heidelberg (2006)
6. Fundulaki, I., Maneth, S.: Formalizing XML access control for update operations. In: ACM SACMAT (2007)

7. Duong, M., Zhang, Y.: An integrated access control for securely querying and updating XML data. In: Australasian Database Conference (2008)
8. ten Cate, B., Lutz, C.: The complexity of query containment in expressive fragments of xpath 2.0. In: Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (2007)
9. Marx, M.: XPath with Conditional Axis Relations. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 477–494. Springer, Heidelberg (2004)
10. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Rewriting regular xpath queries on XML views. In: ICDE (2007)
11. Groz, B., Staworko, S., Caron, A.-C., Roos, Y., Tison, S.: XML Security Views Revisited. In: Gardner, P., Geerts, F. (eds.) DBPL 2009. LNCS, vol. 5708, pp. 52–67. Springer, Heidelberg (2009)
12. Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J., Siméon, J.: Xml path language (xpath) 2.0 (second edition). W3C Recommendation (December 2010), http://www.w3.org/TR/2010/REC-xpath20-20101214/
13. Bravo, L., Cheney, J., Fundulaki, I.: Repairing Inconsistent XML Write-Access Control Policies. In: Arenas, M. (ed.) DBPL 2007. LNCS, vol. 4797, pp. 97–111. Springer, Heidelberg (2007)

Mitigating Collaborative Blackhole Attacks on DSR-Based Mobile Ad Hoc Networks

Isaac Woungang[1], Sanjay Kumar Dhurandher[2], Rajender Dheeraj Peddi[1], and Issa Traore[3]

[1] Department of Computer Science,
Ryerson University, Toronto, Ontario, Canada
iwoungan@scs.ryerson.ca, rpeddi@ryerson.ca
[2] Division of Information Technology
Netaji Subhas Institute of Technology, University of Delhi, India
dhurandher@rediffmail.com
[3] Department of Computer and Electrical Engineering
University of Victoria, B.C., Canada
itraore@ece.uvic.ca

Abstract. A Mobile ad hoc network (MANET) is a collection of mobile nodes that rely on co-operation amongst devices that route packets to each other. From a security design perspective, MANETs have no clear line of defense. This lack of security leads the network accessible to both legitimate network users and malicious attackers. A blackhole attack is a severe attack that can be employed against data routing in MANETs. A blackhole is a malicious node that can falsely reply for any route requests without having an active route to a specified destination and drop all the receiving data packets. The attack may even lead to more devastating damage if two or more blackhole nodes cooperate with each other to launch an attack. This type of attack is known as collaborative blackhole attack. In this paper, a novel scheme Detecting Collaborative Blackhole Attacks (so-called DCBA) for detecting collaborative blackhole attacks in MANETs is introduced. Simulation results are provided, demonstrating the superiority of DCBA compared to Dynamic Source Routing (DSR) and the Bait DSR scheme (so-called BDSR) [1] - a recently proposed scheme for detecting and avoiding collaborative blackhole attacks in MANETs - in terms of network throughput rate and minimum packet loss percentage, when collaborative blackhole nodes are present in the network

Keywords: Mobile ad hoc networks, colloborative blackhole attack, Dynamic Source Routing (DSR).

1 Introduction

MANETs can be described as an infrastructureless wireless networks, in which a group of mobile nodes communicate with each other over a wireless channel in a cooperative manner. In MANETs, mobile nodes can act as both a host and

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 308–323, 2013.

a router while forwarding the packets to other mobile nodes. MANETs are very easy to deploy and are dynamic in nature, hence, these types of networks can be used in places where terrestrial or geographical constraints are present, such as in battlefields, disaster management situations, to name a few.

Since mobile nodes in MANETs communicate over a wireless channel, message security and transmission are indeed a major concern. Routing protocols in MANETs such as DSR and ad hoc demand routing protocol (AODV) were designed without considering any security constraints in MANETs. Thus, AODV-based MANETs or DSR-based MANETs may be vulnerable to several distinct types of attacks, including blackhole attacks[2,3], wormhole attacks, sybil attacks [4], Denial of Message (DoM) attacks, to name a few. In this paper, our focus is on blackhole attacks.

A blackhole attack [5] is an attack where the malicious node (so-called blackhole node) can attract all the packets by using a forged Route reply packet to falsely claim that it has a shortest route to the destination. When the packets reach the blackhole node, they merely disappear. In fact, the blackhole node impersonates the destination node by sending a spoofed route reply packet to the source node that have initiated the route discovery, hence deprives the packets from the source node.

A blackhole node has two fundamental properties. First, it takes advantage of the ad hoc routing protocol such as AODV or DSR to advertise itself as having a valid route to the destination node, even though the route is spurious, with the intention to intercept packets. Second, the node consumes the intercepted packets. Blackhole attacks in MANETs can cause immense harm to the network. When two or more malicious nodes collaborate with each other, i.e. work as a group, the damage can be even worse. This type of attack is known as collaborative blackhole attacks as illustrated in Figure 1.

This paper focuses on detecting collaborative blackhole nodes (i.e. malicious and cooperating malicious nodes) in the network. The remainder of the paper is organized as follows. Section 2 discusses some related work on schemes for

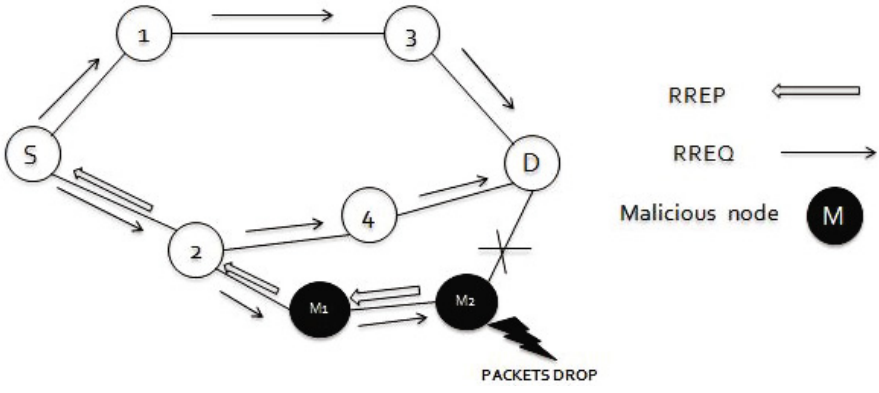


Fig. 1. Example of collaborative blackhole attack in MANET

mitigating blackhole/collaborative blackhole attacks in MANETs. In Section 3, our proposed novel algorithm to detect and isolate blackhole nodes in MANETs is discussed. In Section 4, the performance evaluation of our scheme is presented. Finally, Section 4 concludes our work.

2 Related Work

Several proposals in the literature have dealt with detecting and/or avoiding malicious blackhole nodes in MANETs. Few recent representative ones are as follows.

In [6] Deng et al. proposed a solution which asks every intermediate node to include the information on the next hop to destination in its route reply (RREP) packet when the intermediate node replies to the route request (RREQ) packet. While receiving the RREP, the source node does not transmit the data packets to the intermediate node immediately. Rather, based on the receiving information on the next hop, the source node sends a FutherRequest (FRq) to the next hop node to ask whether this node has a valid route to the destination. The source node receives a FutherReply (FRp) message from the next hop, which includes the check result. If the answer is yes in the FRp message received by the source node , the route is built and the source node transmits the data. If the answer is no, the source node sends an Alarm Packet to alert other nodes in the network about that fact. However, this method has some drawbacks, namely (1) the process of checking the validity of RREP from an intermediate node through FRq and (2) FRp messages obviously leads to some overhead in the network. these issues were not addressed by the authors [6]. Moreover, their proposed algorithm only addressed single blackhole attacks, and cannot mitigate cooperative blackhole attacks.

In [7], Weerasinghe and Fu proposed a solution for preventing cooperative blackhole attacks in MANETs. They proposed a slightly modified AODV protocol in which the data routing information table (DRI) and cross checking are introduced. More precisely, each node maintains its DRI table and used it to identify the misbehaving nodes in the network. The DRI table keeps track of whether or not a node has transferred the data to its neighbours. Nodes are also crosschecked by using the FREQ and FREP packets. However, the overhead caused by the FREQ and FREP packets appeared to increase the end-to-end delay in the network.

In [1], Tsou et al. proposed a scheme called Bait DSR (BDSR) to detect and avoid blackhole attacks in MANETs. According to their solution, the source node sends a bait RREQ packet before the actual routing process takes place. This is used to bait the malicious nodes, which then reply to the bait RREQ packet. Based on this response, malicious nodes are then detected using a reverse tracing technique.

In [8], Marti et al. proposed a watchdog and pathrater scheme to detect malicious nodes present in a MANET. The watchdog method identifies the malicious nodes in the MANET by eavesdropping on the transmissions of the next hop

node. Watchdog compares each overheard packet with the packets in the buffer, which contains the packets recently sent by a node. If there is a match between the packets, the node removes the packets from the buffer; otherwise it increments a failure tally for the neighbouring node. If a packet has remained in the buffer for longer than a certain timeout period. A node is identified as a malicious node if the tally exceeds a certain threshold bandwidth. In this situation, the source node is notified about this malicious node. The pathrater method then helps in finding the routes that do not contain those malicious nodes. In this scheme, each node keeps track of the trustworthiness rating of every known node. The pathrater chooses the shortest path if there are multiple paths to the destination. The main drawback of this method is that it might not detect malicious node in the presence of limited transmission power, false behaviour or partial dropping.

In [9], William et al. proposed a scheme (so-called REAct system) for detecting malicious nodes in MANETs. Their scheme is made of three phases: audit, search and identification. The audit phase verifies the packets forwarded from the audit node to the destination node. The source node will choose an audit node to use bloom filter in order to generate a behavioural proof. The source node also uses bloom filter to produce a behavioural proof and compare it with against the proof produced by the bloom filter generated by the audit node. As a result of this comparison, the segment that has the malicious node is identified. However, this method has an oblivious drawback, i.e. it can identify the blackhole attack only after the damage has been done to the network.

In [10], Raj et al. proposed a scheme called DPRAODV to detect and isolate blackhole attacks in MANETs. In their approach, whenever the source node receives a RREP packet, the packet first checks the value of the sequence number in its routing table and does an additional check to find whether the RREQ sequence number is higher than a specified threshold value. This threshold value is dynamically updated at every predefined time interval. If the value of the RREP sequence is higher than the threshold value, that particular node is identified as blackhole node, which is blacklisted and an ALARM packet is sent to all other nodes in the network so that the RREP packet originated from that malicious node is discarded and the routing table for that node is not updated. The ALARM packet has the address of the malicious node as a parameter so that, the neighbours know that the RREP packet from the node is to be discarded. However, this algorithm suffers from excessive overhead due to the fact the threshold value has to be updated at every time interval and special ALARM control packets should be handled.

In [11], Tamilselvan et al. proposed an AODV-based protocol for preventing blackhole attacks in MANETS. In their approach, the source node awaits until the intermediate node replies with the next hop details in the RREP packet. Whenever the source node receives the RREP packets, it records the sequence number along with the time the packet arrived in a collect route reply table (CRRT). After receiving the RREP packets, the source nodes calculates the timeout value for each RREP packet, and then checks the CRRT for any repeated

next hop nodes. The route with the repeated next hop node is then considered as safe and can be used to route the packets. However, if there is no path with repeated next hop nodes, the algorithm chooses a random path from the CRRT, which may lead to blackhole attack in the network.

In [13], Buchegger et al proposed a trust based solution named Cooperation Of Nodes, Fairness In Dynamic ad hoc Networks (CONFIDANT) to avoid blackhole attack in MANETs. Their solution is similar to the solution proposed in [8] with a trust manager. The trust manager evaluated the events reported by the watchdog and issues a alarm packet to warn other nodes about the malicious nodes. Each node in the network monitors the behaviour of its neighbour and notifies the reputation system in the event of any suspicious activity. The The reputation system maintains a black-list of nodes at each node and shares them with nodes in the friends-list. If the node's trust value drops below the threshold value the CONFIDANT protocol deletes all routes containing the misbehaving node from the path cache and doesn't allow the node to participate in forwarding the data packets.

Most of the above-discussed solutions to avoid blackhole attacks in MANETs are used to detect single blackhole attacks. In this paper, a DSR- based protocol is presented to mitigate collaborative blackhole attacks in MANETs. Unlike other schemes for preventing blackhole attacks, in which the malicious nodes are identified only after the actual routing process started, in our solution, the blackhole nodes or collaborative blackhole nodes are identified before the actual routing process takes place.

3 Proposed Mitigation Scheme

In this section, a modified version of the DSR protocol (our so-called DCBA) is proposed to find a secure route between the source and destination nodes and isolate the malicious blackhole nodes in MANETs. Our approach merges the advantage of proactive detection in the initial stage and reactive mechanism at later stages if the proactive detection approach fails to identify the malicious blackhole nodes. Consequently, our mechanism is different to other methods that just use a reactive approach that would suffer a blackhole attack in its initial stage. In our proposed algorithm, malicious nodes are identified by means of our so-called suspicious values of nodes. A suspicious value is an important parameter to judge the behavior of a node (i.e. either it is malicious or non-malicious). As a source routing protocol, DSR can identify the addresses of all the nodes in a routing path once the source node has receive the RREP message in response to a RREQ message. However, the source node itself cannot identify exactly which intermediate node has the route information to the destination node and the reply RREP. This situation can result to the source node sending packets to the fake shortest path claimed by a malicious node (among available existing ones if any), yielding a blackhole attack that causes packets loss. However, it is difficult to identify which malicious node(s) generated the packets loss.

Our DCBA protocol Figure 2 combines a modified DSR and the BDSR protocol [1], to yield a strong method for detecting collaborative blackhole attacks in

MANETs. Indeed, the packet format of the RREP message in DSR is modified as follows. In the RREP, the reserved field is changed to the RREP initiator address field. The latter will store the address of the node that replies to the RREP. This RREP initiator address field can help tracing the intermediate node that claimed it has the shortest route to the destination node. To achieve this goal, the concept of suspicious value attached to anode is introduced, which is described as follows.

Algorithm 1 DCBA

Require: DSR Protocol (MANETs)
RREQ: Route request packet
RREP: Route reply packet
SN:Source node
DN: Destination node
ACK: Acknowledgement packet
IN: Intermediate node

```
1: SN broadcasts the RREQ packet
2: SN receives RREP
3: if RREP is from any IN then
4:     SN checks the RREP packet for the address of the node that initialized the RREP
5:     SN checks the Suspicious value of the intermediate node that initialized RREP
6:     if Suspicious value is below a threshold value then
7:         Consider the route to be safe and start routing the data packets
8:     else
9:         Mark the node as malicious node and build a blackhole list
10:    end if
11: else
12:    Start routing data packets
13:    if The packet delivery ratio is below the threshold value then Consider route to be
   danger
         and restart the routing process
14:    end if
15: end if
```

Fig. 2. DCBA Algorithm

In our proposed method, each node has its own suspicious value, which is based on the abnormal difference observed between the routing messages transmitted from the node. Suspicious values for each node are stored in a table (so-called suspicious values table). This table for each node is updated periodically after a certain time interval. Whenever the source node receives the route reply (RREP) packet in reply to the route request (RREQ) packet, it checks the RREP packet for the address of the node that initiated the RREP packet. The source node checks the suspicious value of the node that initialized the RREP packet. If this value is higher than the threshold level, then the node is considered as malicious

and its address is stored in a blacklist table, preventing that node to further participate in the routing process. The threshold value is variable and can be adjusted depending on the performance of network.

In general, if an intermediate node is not the destination node, and it never broadcasts a RREQ, but forwards a RREP for the route, then its suspicious value is increased in the suspicious value table. Only the source node has the right to update or modify the suspicious value table, thus whenever the source node realizes that a nodes suspicious value is to be increased, it will notify every other node in the network in order to have them update its suspicious values table. When the suspicious value of a node reaches the prescribed threshold value, it is considered as a malicious node.

3.1 Routing Mechanism

Whenever the source node wants to send some data to the destination node, it initiates the route discovery process. In this process, the source node broadcasts

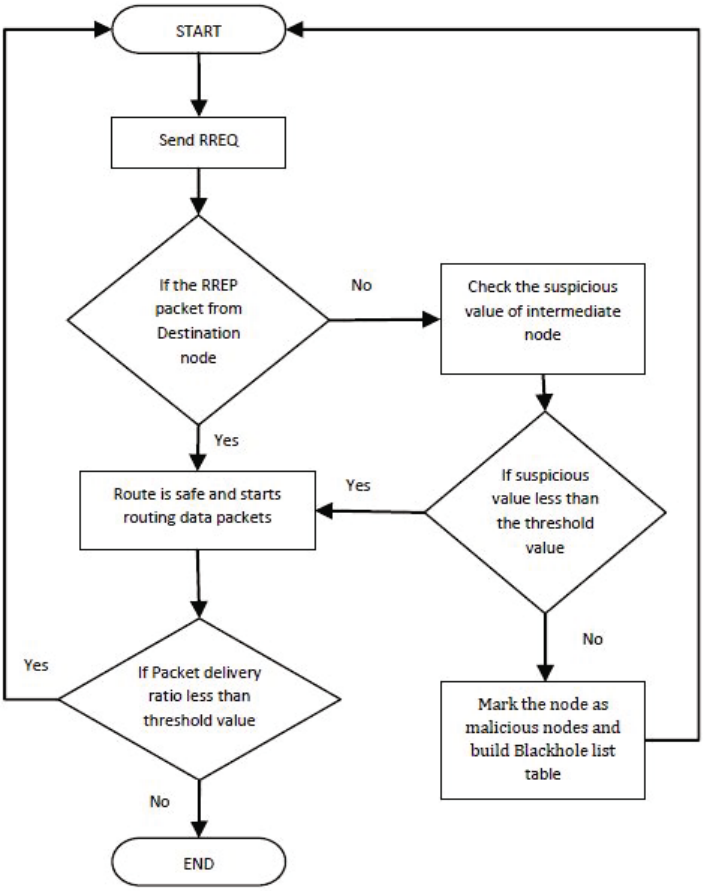


Fig. 3. DCBA algorithm flowchart

the Route Request (RREQ) packet. All the intermediate nodes that receive this RREQ packet check their routing table for the routing information to the destination node. If the intermediate node has the routing information to the destination, it will reply with a Route Reply (RREP) packet to the source node. When the source node receives the RREP packet, it checks the RREP for the address of the node that initiated the RREP packet using the RREP imitator address field in the RREP packet. Then, the source node checks the suspicious value of the node that initiated the RREP. If the suspicious value of that node is higher than the specified threshold level, then the source node sends an alarm message to all other nodes, indicating that there is a malicious node and updates the blacklist table with the address of that malicious node. If the suspicious value is below the specified threshold value, the source node will start routing the data packets. If the destination node detects that the packet delivery ratio drops to a threshold obviously after the route had been build, the detecting mechanism will be triggered again to avoid blackhole nodes that may have not been detected Consequently, our mechanism can keep protecting and reacting immediately. The flow chart of our DCBA algorithm is depicted in Fig. 3.

3.2 Correctness of DCBA

Lemma 1: *DCBA algorithm converges to a correct (i.e. safe) routing path for packet delivery in a finite time when the RREP received is from an intermediate node.*

Whenever there is a need to route the data packets, the source node initializes the route discovery process and initiates the timer. On receiving the Route REQuest (RREQ) packets, the intermediate node in the network replies with a Route REPly (RREP) packet if it has a route to the destination node in its routing cache or relays the RREQ packets to the neighbouring nodes. The RREP packet contains the path to the destination, the RREP initiator address and the hop count. On receiving the RREP packet, the source node analyzes it by checking the suspicious value of the intermediate node. After analyzing all received RREP packets, the source node selects the best route to the destination i.e. the route with less suspicious value and least number of hops, then updates the routing table with the routing information. If the suspicious value of an intermediate node is found to be more than the prescribed threshold value, the source node immediately marks that intermediate node as malicious, adds it to a blackhole list table, then it notifies all the nodes in the network about that malicious node. The source node also discards all RREQ packets and RREP packets present in the network after a certain timeout

Lemma 2: *DCBA algorithm converges to a correct (i.e. safe) routing path for packet delivery in a finite time when the RREP received is from a destination node.*

On receiving the RREP packet from the destination node, the source node trusts the RREP packet since it originated from the destination node, then starts routing the data packets. The destination node replies to the source node if there is an

abnormal difference between the sent and received packets i.e. the packet delivery ratio. After receiving a reply from the destination node, the source node initializes the reactive approach by stopping immediately the transmission of the data packets. It also deletes all the routing information that are related to the current route then notifies all the nodes to update their routing tables. After the routing tables have been updated by all nodes, the source node again initializes the route discovery process to identify a fresh and safe route to transfer the data packets.

4 Performance Evaluation

GloMosim [14] is the simulation tool used to implement the proposed DCBA scheme for identifying the collaborative blackhole nodes. The simulation parameters and experimental variables are inherited from [5]. These are captured in Table 1. A MANET with 50 nodes is designed, and the choice of malicious nodes in the network is random. A source node and a destination node are selected, and about 1000 data packets of 64 bytes each are transmitted from source to destination.

The proposed DCBA scheme was compared against the normal DSR (denoted DSR) scheme and the BDSR scheme [1], chosen as benchmark, under two varying simulation parameters: the pause time and the percentage of malicious nodes, on the basis of the following performance metrics:

- **Packet Delivery Ratio:** This metric represents the ratio between the number of packets originated by the application layer sources and the number of packets received by the sink at the final destination. The greater the packet delivery ratio, the better the performance of the network will be.
- **Average End-to-End Delay:** This metric is the average time taken by the packet to reach the destination. This includes the time from generating the packet from the source node up to the reception of the packet by the destination node. It is expressed in seconds. This metric includes the overall delay of the network including buffer queues, transmission time and induced delay due to routing activities. The lower the value of the end-to-end delay, the better the performance of the network will be.
- **Network throughput:** This metric represents the average rate of successful message delivery over a communication channel. It can be measured as bits per second (bps), packets per second (pps) or packet per time slot.
- **Routing Overhead Ratio:** The routing overhead can be defined as the ratio of the amount of routing related control packets transmitted to the amount of data packets transmitted by the application traffic.

The effect of the packet delivery ratio (PDR) on the percentage of malicious nodes in the network is first investigated. The results are captures in Figure 4. It can be observed that DSR suffers heavy loss in packets in the presence of blackhole nodes. This can be justified by the fact that DSR does not have any intrinsic detection and prevention mechanism to prevent blackhole attacks. Also the BDSR scheme uses a fake RREQ technique to find the blackhole attack, it can

suffer packet loss if the malicious node does not reply to the RREQ packet. When varying the percentage of malicious nodes from 0% to 40%, DCBA generates a higher and consistent PDR compared to BDSR scheme, even in the presence of collaborative blackhole nodes.

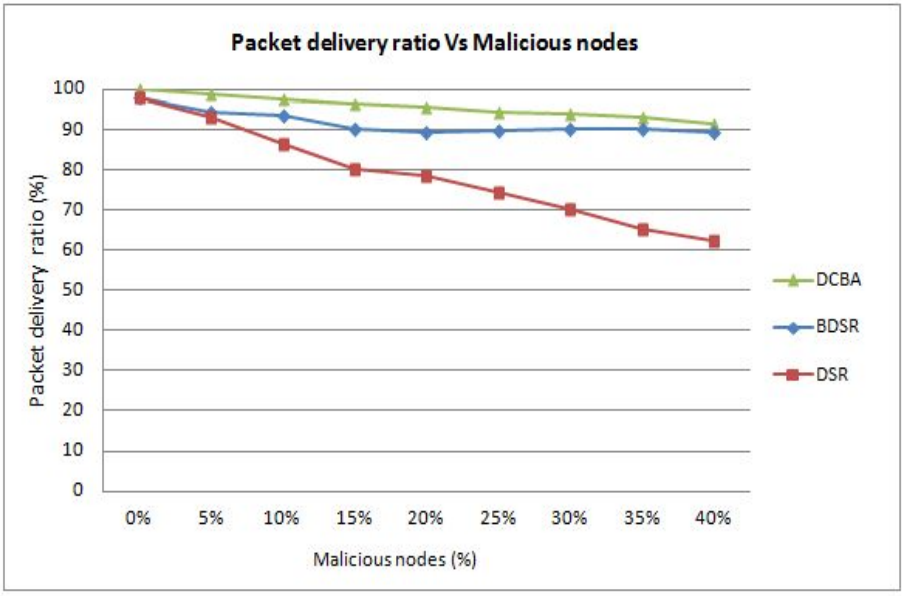


Fig. 4. DCBA algorithm - Packet delivery ratio Vs Malicious nodes

The effect of the packet delivery ratio on the pause time is also investigated, and the results are depicted in Figure 5. It can be observed that the packet delivery ratio drops as the pause time is increased. It can also be observed that our DCBA scheme generates higher packet delivery ratio compared to the BDSR scheme and the normal DSR protocol even in the presence of the collaborative blackhole nodes. Finally, it can be observed that the packet delivery ratio for the normal DSR protocol ranges between 94% and 66%. Our protocol DCBA improves the situation by increasing the packet delivery ratio by more than 20%. This can be justified by the fact that the normal DSR does not have any built-in security mechanism.

The second performance metric used in the analysis of our solution is the network throughput. The effect of the network throughput when the percentage of malicious nodes in the network increases is investigated. The results are captured in Figure 6.

First, it can be observed that, DSR heavily suffers from the collaborative blackhole attacks since the protocol does not have any mechanism to prevent these attacks. Moreover, the throughput of DSR goes down under 300bps as the number of blackhole nodes in the network increases from 0% to 40%. Second, the throughput for the BDSR scheme ranges between 520bps and 480bps

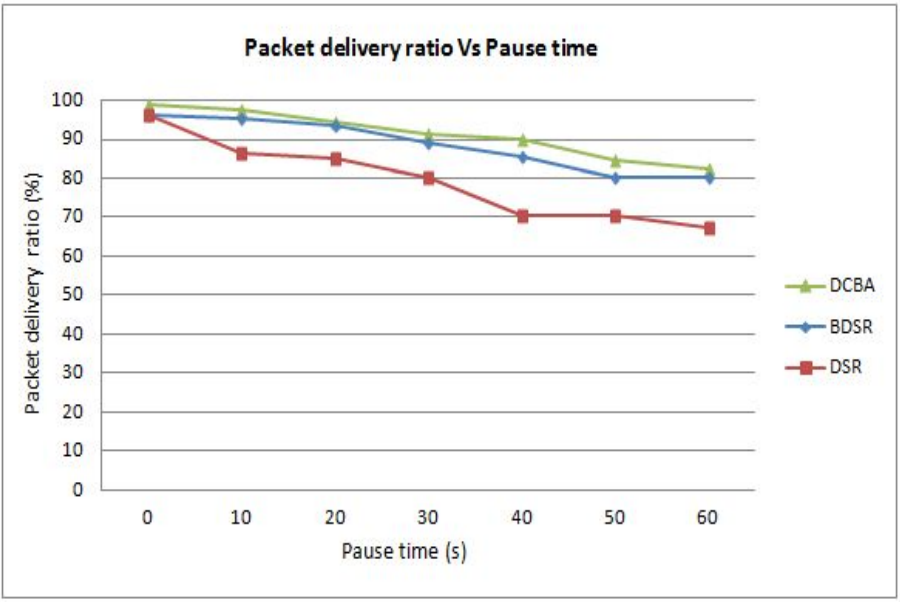


Fig. 5. DCBA algorithm - Packet delivery ratio Vs Pause time

as the number of malicious nodes increases. Thirdly, our protocol generates a higher throughput than the other two protocols. This is due to the fact that our scheme prevents packet drops by malicious nodes using the proactive mechanism. Even with the 40% of blackhole nodes, our protocol produces a throughput of 590bps. Furthermore, it can be observed that the normal DSR protocol under collaborative blackhole attack has the lowest throughput and BDSR also has lesser throughput when compared to that of the DCBA scheme under blackhole/collaborative blackhole attacks.

Next the effect of the network throughput on the pause time is also investigated, and the results are depicted in Figure 7. As the pause time increases, the paths between the source node and the destination node lasts longer and becomes more stable. Therefore, the data packets transmitted along transient routes (resulting from quick node movement) decreases, thus reducing the overall throughput. First, it can be observed that the network throughput under normal DSR protocol decreases as the pause time increases. Secondly, it can be observed that the throughput under the BDSR scheme is higher than that obtained with the normal DSR scheme and our DCBA protocol has the higher throughput compared to that of DSR and BDSR. This can be justified by the fact that DCBA protocol is capable of mitigating the blackhole attacks in the network.

The third performance metric used in the analysis of our solution is the routing overhead ratio in the network. The effect of the routing overhead ratio on pause time is depicted in Figure 8. First, it can be observed that the DSR protocol

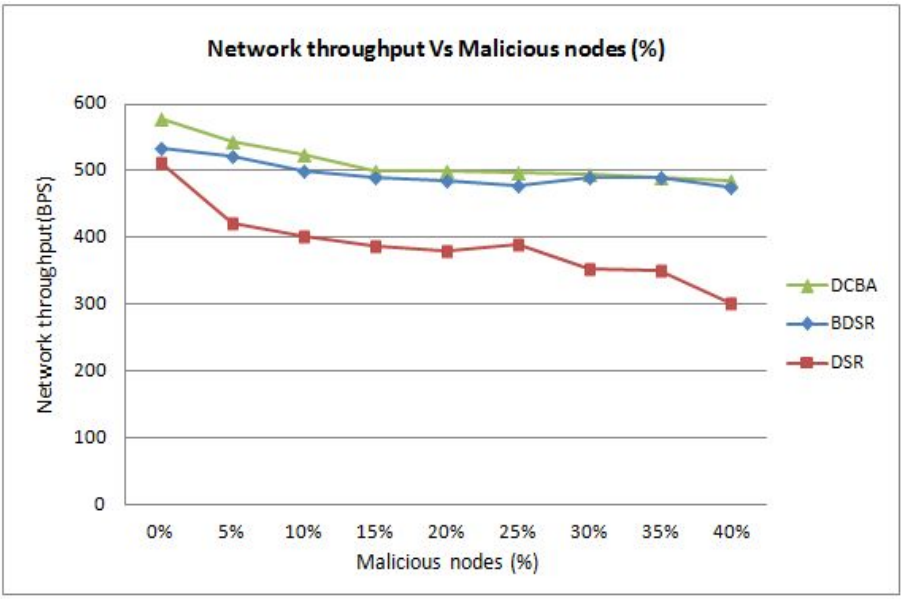


Fig. 6. DCBA algorithm - Network throughput Vs Malicious nodes

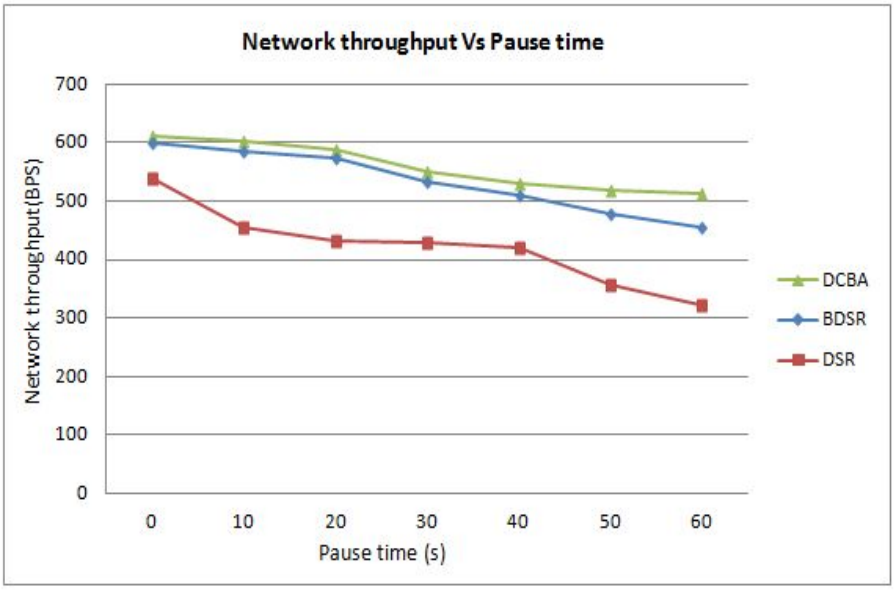


Fig. 7. DCBA algorithm - Network throughput Vs Pause time

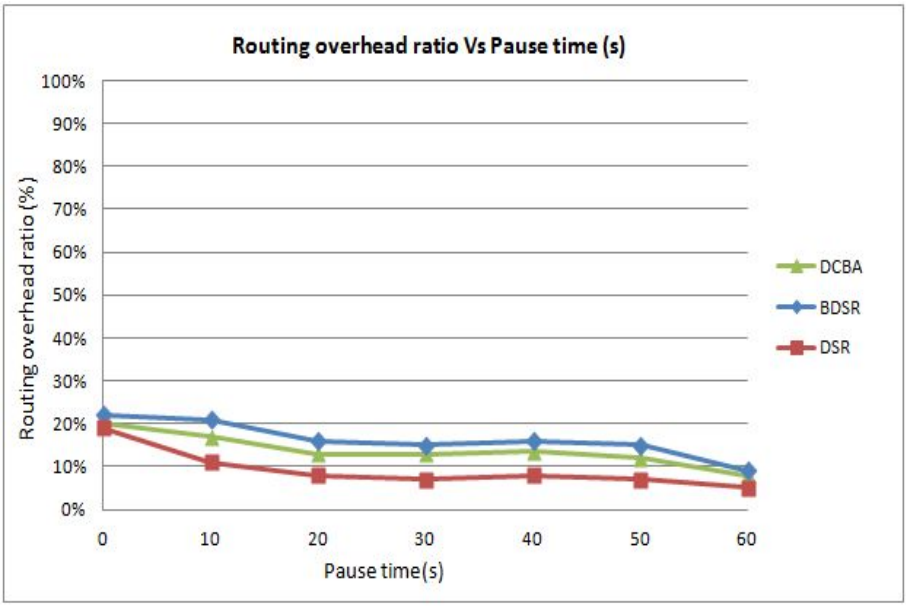


Fig. 8. DCBA algorithm- Routing overhead(%) Vs Pause time

routing overhead decreases as the pause time increases. This is due to the fact that the increase in pause time causes the attacker to establish a more stable path between the source and destination. As paths become more stable, the required number of routing related packets reduces. Secondly, the routing overhead ratio for the BDSR protocol is higher than that of the DSR protocol because the BDSR scheme uses more RREQ packets to find the secure route in the presence of blackhole nodes. Thirdly, DCBA's routing overhead is greater than that of the DSR scheme and less then that of the BDSR scheme because our protocol does not need the use of fake RREQ packets as the BDSR does.

The effect of the routing overhead ratio on the percentage of malicious nodes is depicted in Figure 9. First, the DSR protocol introduces the lowest overhead due to the fact that it does not use any additional requests for finding secure routes. Also the routing overhead ratio for DSR decreases as the number of malicious node increases. This is due to the fact that the presence of more malicious node in network causes immediate reply to the route requests, which in turn causes less overhead. Second, the routing overhead of the BDSR protocol is greater than that of the DSR protocol since BDSR uses the extra RREQ packets to bait the blackhole nodes.Third, DCBA introduces a lower overhead compared to BDSR and more overhead compared to DSR. This might be due to the fact that our solution uses normal RREP packets header to check the suspicious value of the node.

The impact of the number of malicious black hole nodes on end-to-end delay is depicted in Figure 10 . It can be observed that the delay in DSR decreases when the percentage of blackhole nodes increases. This is justified by the fact that an

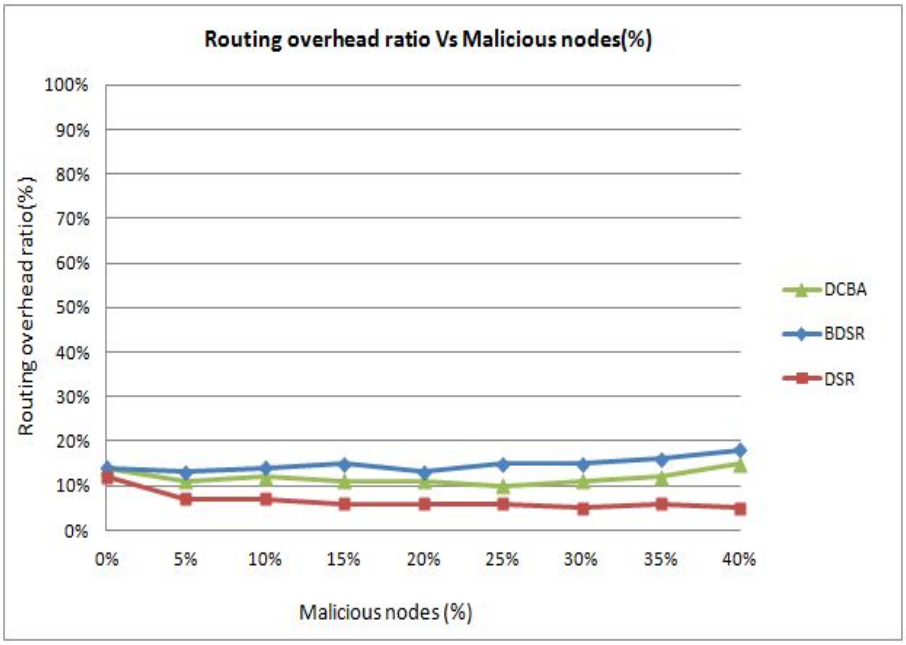


Fig. 9. DCBA algorithm- Routing overhead(%) Vs Malicious nodes

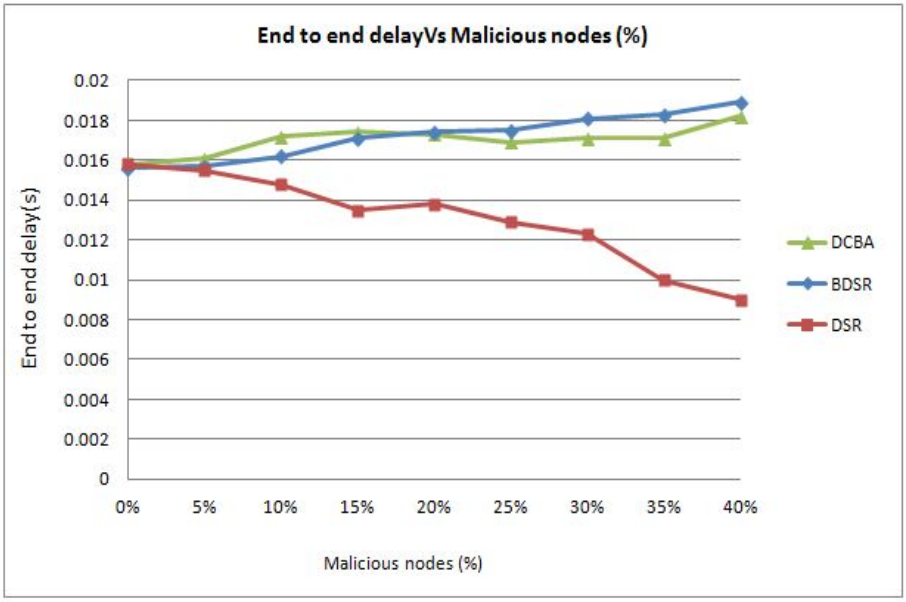


Fig. 10. DCBA algorithm- End-to-end delay Vs Malicious nodes

increase in number of malicious nodes means that the source will have to find more routes between source to destination in less time because more blackholes reply quickly for the route requests. Secondly, the delay for the BDSR protocol and DCBA protocol increases with the increase in malicious nodes since it has to avoid more malicious nodes when it tries to find out secure route from source to destination.

5 Conclusion

In this paper, we proposed a scheme (so-called DCBA) to identify and mitigate blackhole/collaborative blackhole attacks in MANETs. Simulation results showed that (1) the original DSR heavily suffers from blackhole/collaborative blackhole attacks in terms of network throughput and packet delivery ratio, (2) the proposed scheme outperforms both the DSR and BDSR schemes in terms of network throughput rate and minimum packet loss percentage.

References

1. Tsou, P.C., Chang, J.M., Lin, Y.H., Chao, H.C., Chen, J.L.: Developing a bdsr scheme to avoid black hole attack based on proactive and reactive architecture in manets. In: 2011 13th International Conference on Advanced Communication Technology (ICACT), Phoenix Park Gangwon-Do, Korea (South), pp. 755–760 (2011)
2. Tseng, F.H., Chou, L.D., Chao, H.C.: A survey of black hole attacks in wireless mobile ad hoc networks. Human-centric Computing and Information Sciences 1, 1–4 (2011)
3. Hu, Y.C., Perrig, A., Johnson, D.B.: Rushing attacks and defense in wireless ad hoc network routing protocols. In: Proceedings of the 2nd ACM Workshop on Wireless Security, WiSe 2003, pp. 30–40. ACM, New York (2003)
4. Khokhar, R.H., Ngadi, A.N., Mandala, A.: A review of current routing attacks in mobile ad hoc networks. International Journal of Computer Science and Security 3, 18–29 (2008)
5. Royer, E.M., Toh, C.K.: A review of current routing protocols for ad-hoc mobile wireless networks. IEEE Personal Communications 6, 46–55 (1999)
6. Deng, H., Li, W., Agrawal, D.P.: Routing security in wireless ad hoc networks. IEEE Communications Magazine 40, 70–75 (2002)
7. Weerasinghe, H., Fu, H.: Preventing cooperative black hole attacks in mobile ad hoc networks: Simulation implementation and evaluation. In: Proceedings of the Future Generation Communication and Networking, FGCN 2007, vol. 02, pp. 362–367. IEEE Computer Society, Washington, DC (2007)
8. Marti, S., Giuli, T.J., Lai, K., Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks. In: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, MobiCom 2000, pp. 255–265. ACM, New York (2000)
9. Kozma, W., Lazos, L.: React: resource-efficient accountability for node misbehavior in ad hoc networks based on random audits. In: WISEC, pp. 103–110 (2009)

10. Raj, P.N., Swadas, P.B.: Dpraodv: A dynamic learning system against blackhole attack in aodv based manet. International Journal of Computer Science Issues, IJCSI 2, 54–59 (2009)
11. Tamilselvan, L., Sankaranarayanan, V.: Prevention of blackhole attack in manet. In: Proceedings of the 2nd International Conference on Wireless Broadband and Ultra Wideband Communications, AUSWIRELESS 2007, pp. 21–28. IEEE Computer Society, Washington, DC (2007)
12. Ramaswamy, S., Fu, H., Sreekantaradhya, M., Dixon, J., Nygard, K.: Prevention of cooperative black hole attack in wireless ad hoc networks. In: Proceedings of Intl Conf. on Wireless Networks, Las Vegas, Nevada, USA, pp. 570–575 (2003)
13. Buchegger, S., Le Boudec, J.Y.: Performance analysis of the confidant protocol. In: Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking & Computing, MobiHoc 2002, pp. 226–236. ACM, New York (2002)
14. Glomosim, `http://pcl.cs.ucla.edu/projects/glomosim/` (last accessed: August 24, 2012)

QoS Aware Adaptive Security Scheme for Video Streaming in MANETs

Tahsin Arafat Reza and Michel Barbeau

School of Computer Science, Carleton University,
1125 Colonel By Drive, Ottawa ON K1S 5B6, Canada
treza@connect.carleton.ca, barbeau@scs.carleton.ca

Abstract. Real-time video streaming is delay sensitive. It has minimum bandwidth and QoS requirements. Achieving target QoS for video streaming is challenging in a decentralized and self-organized MANET. Cryptography algorithms offer confidentiality of shared data, but they have computation cost. Our work addresses the issue of delay overhead caused by the introduction of cryptography that directly affects video streaming performance. Our proposal is motivated by possibilities of adaptive security and multimedia service. We make an effort to identify why, when and how to deploy adaptation. We propose QaASs (QoS aware Adaptive Security scheme), an adaptive mechanism that counters the effect of delay overhead by adapting cryptography and multimedia properties, providing QoS while maintaining a required level of security. We evaluate our proposal through implementation and simulation.

Keywords: Ad hoc Network, QoS, Adaptive Security, Video Encryption, Elliptic Curve.

1 Introduction

A Mobile Ad hoc Network (MANET) is a communications network where there is no fixed infrastructure or central authority. The nodes are self-organized and communicate with each other directly or through intermediate nodes. Nodes act as hosts and routers. No static topology is guaranteed. There is a growing interest for real-time video streaming in MANETs. Possible usages are remote surveillance, environmental or wildlife monitoring, rescue operations, telemedicine in adverse environments, collaborative unmanned remote exploration, ad hoc network of UAVs (Unmanned Arial Vehicles) and UWVs (Unmanned Under Water Vehicles). In a VANET (Vehicular Ad hoc Network), peers can engage in video conference as well as stream media in an ad hoc manner.

In a computer network, security measures are deployed as a protection against malicious attacks or intentional faults that disrupt regular operations and unauthorized access to resources and information. The physical construction and functional characteristics of a MANET make it vulnerable and susceptible to malicious attacks. Absence of infrastructure, broadcast nature of wireless transmission, sole dependency on wireless links, dynamic topology, and multihop routing have been identified as the primary features that make MANETs vulnerable

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 324–340, 2013.

to malicious attacks [8]. Security techniques for infrastructure-based networks are not applicable to MANETs. For example, the use of a unique certification authority (CA) is against the core concept of infrastructure-less networks. *Eavesdropping*, *Tunneling*, *Spoofing*, *Rushing*, *Wormholes*, *Black holes* and various DoS (Denial of Service) attacks [8] are examples of security setbacks in MANETs. *Confidentiality* is a must have requirement for distributing and sharing sensitive information [36]. Confidentiality refers to protection against unauthorized disclosure of information. Cryptography provides security for digital contents. Real-time video streaming is delay sensitive, involves encoding and decoding and has minimum bandwidth and QoS (Quality of Service) requirements [27]. QoS is a set of service requirements (e.g., delay, data rate and error correction) to be met by the network while transporting a packet data stream. Achieving target QoS for video streaming is challenging in an unpredictable MANET. Cryptography algorithms could be computationally intensive. Computation overhead introduced by cryptography operations may cause additional delay to video streaming which could directly influence playback experience.

Our work addresses the issue of delay overhead caused by the introduction of cryptography that directly affects video streaming performance. A MANET can be composed of a diverse range of devices with different computation capabilities. The performance of a computationally intensive cryptography process varies depending on the available system resource (e.g., memory and number of running threads). A cryptography process may introduce additional, yet unavoidable delay overhead. If a traffic source knows the capability of a target device, e.g., the throughput of a cryptography process, then it can infer appropriate cryptography parameters that do not cause a performance bottleneck. Furthermore, it may be possible to control multimedia traffic, thus the amount of data to be processed by a cryptography processor. Traffic load influences network latency as well as congestion, thus packet delivery ratio. By adjusting multimedia parameters, it is possible to control the overall delay as well as provide QoS. The receiver of multimedia service can provide periodic feedback to the source with information such as transmission delay, delay jitter, effective frame rate and frame loss ratio. Hence, an adaptive mechanism that trades off between security and QoS parameters is a feasible solution to the addressed problem.

We propose QaASs (QoS aware Adaptive Security scheme), a runtime adaptive mechanism that counters the delay overhead by adapting cryptography and multimedia properties, providing QoS while maintaining a required level of security. The mechanism is designed around a cryptography delay threshold value and considers cryptography process throughput and delay, and video reception rate. QaASs defines *why*, *when* and *how* to deploy adaptation. We demonstrate the effectiveness of our proposal by presenting a number of service scenarios demanding different requirements with results confirmed with a 95% confidence level [7].

The rest of the paper is organized in four sections. In Sect. 2, we present a literature review related to our problem of interest. Relevant background information are detailed in Sect. 3. In Sect. 4, we describe our proposal. Simulation

and results are documented in Sect. 5. Section 6 concludes the paper and outlines future work in this direction.

2 Related Work

The notion of adaptive security takes root in autonomic security management [13]. Shnitko [32] identified adaptive security as a problem of optimal control of an object whose state is influenced by a set of adaptable factors and environment parameters. He stressed on the necessity of adaptive approaches for information security in order to cope with the uncertainty of the environment. A class of adaptive security approaches focus on defying the impact of performance degradation and resource exhaustion as a result of security provisioning. [42], [25], [4] and [3] used adaptive security for multimedia QoS, [47], [37], [13] and [23] for energy efficiency, and [3] and [30] for computing resource efficiency. Nijim and Ali [21] proposed an adaptive security approach to enhance disk response time. [34] proposed an adaptive security method for time-critical DBMSs (Database Management System) by partially compromising security for improved timeliness. Preda et al. [28] described an adaptation technique for policy deployment and dynamic refinement of contextual security policies where security devices are unaware of context semantics. Zou et al. [49] proposed the use of adaptive security to create an intelligent firewall to trade off between security and performance. [4] and [37] used AHP (Analytical Hierarchical Process) for modeling an adaptive security solution. [49] and [2] used a fuzzy logic-based approach. Alia et al. [3] presented a Component Composition Selection problem based adaptation model enabling fine-grained trade-offs between QoS and security. He et al. [13] presented a DSL (Domain Specific Language) [40] to describe security adaptation policies for self-protection with emphasizes on runtime adaptation.

Cryptography can be applied to real-time streaming video in several manners. Encryption can be employed in the transform domain, within the video encoder. The DC component and motion vectors are encrypted [11] [20]. Format compliance is a key issue for this approach. The second approach is post compression encryption, where encoded video frames are encrypted individually. The third approach is encrypting packet payload of the multimedia streaming protocol, e.g., RTP [26]. Spanos and Maples [35] were among the first to introduce selective encryption by encrypting only the I-frames of MPEG coded video. Kamphenkel and Blank [18] proposed an adaptive security model called Intelligent Network (IN) to address the issue of delay overhead caused by cryptography. IN offers security and congestion aware path selection and allows separate streams in different classes of reliability. Vaidya et al. [39] proposed a secured multipath traffic allocation technique for VoIP in MANETs. The core bitstream of G.727 [17] coded data is transmitted over the primary path (fail-safe and higher data rate) of and enhancement bitstream over the secondary path. In a similar work, Gibson et al. [11] proposed selective encryption for scalable speech coding (SNR) [9] over MANETs. SECMPEG [20], proposed by Meyer and Gadegast, selectively encrypts DC components, I-blocks and motion vectors, sequence and slice headers of MPEG video according to security level. Tang incorporated

cryptography at the video coding level to achieve compression and encryption in one step [38]. Tang's work was among the firsts that oppose what used to be a common belief that existence of spatial and temporal correlation in video coding, e.g., MPEG, makes cryptography difficult to be applied at the coding level [19]. However, the technique suffers from reduced compression rate and is not suitable for highly sensitive video data. Iqbal et al. [15] proposed a slice-based encryption technique for MPEG-4 H.264/AVC [12] video. MPEG-21 [16] gBSD (generic Bitstream Syntax Description) is used as a metadata descriptor for the compressed bitstream which can be used for adapting compressed video data according to the network condition or application requirements. Mahmud et al. [4] presented an adaptive security architecture for IP-based air-ground communications. A multilevel QoS policy based technique aims to maintain a trade-off between performance and security policies and allocates the network resources accordingly.

3 Background

The goal of a multimedia adaptation technique is to ensure or enhance QoS. Transcoding, simulcast and HTTP Live Streaming are examples of key multimedia adaptation techniques [10] [41]. Availability of variable resolution and bit rate streams are the core ideas of these techniques. MCD (Multiple Description Coding) and H.264/SVC (Scalable Video Coding) offers varying quality for a single coded video [10] [31]. Computation cost and infrastructure dependency make these techniques not suitable for delay sensitive streaming and MANETs.

MPEG-4 H.264/AVC (Advanced Video Coding) [12] is a video coding standard of the ITU-T Video Coding Experts Groups and ISO/IEC Moving Pictures Experts Group (MPEG). The H.264/AVC standard aims to provide network-friendly representation of video data for both conversational (e.g., videoconference) as well as non-conversational (e.g., video-on-demand) applications. A H.264/AVC frame may be composed of multiple slices. There are three main types of slices, namely *I*, *P* and *B*-slice. An I-slice has the macroblocks coded using *intra* prediction. The P-slice extends properties of the I-slice and also contains macroblocks coded using *inter* prediction with at most one motion-compensated prediction signal per prediction block. In addition to properties of the P-slice, the B-slice allows two motion-compensated prediction signal. A sequence of P and B-frames and their reference I-frame compose a Group Of Pictures (GOP). Video can be encoded at different frame rates (frames per second or FPS and abbreviation of unit of measurement is *fps*), e.g., 30, 20, and 10 fps. Interested readers may refer to [12] and [43] for details.

In [29], we presented a framework for video simulation over a MANET. Through evaluation, we have identified MANET components and demonstrated their effectiveness. We used MP-OLSR (Multipath OLSR) [46] and 802.11e (EDAC) [14] for stateless admission control at the MAC layer. In [29], we evaluated how cryptography affects H.264/AVC coded video streaming for different cryptography schemes and traffic load. The results complement Yau et al.'s

work [45] that showed that AES with 128-bit key and 256-bit key has no significant performance difference. However, ECC (Elliptic Curve Cryptography) throughput (processed data as a function of time) performance significantly decreases with increasing curve size. Average cryptography delay per frame for ECC with a 521-bit curve is 147 ms compared to 33 ms of ECC with a 256-bit curve and sub 2 ms of AES with 256-bit key.

We are particularly interested in using ECIES (Elliptic Curve Integrated Encryption Scheme), for encrypting multimedia contents. The key advantages of ECC are use of binary polynomial that does not require integer multiplication. This reduces hardware implementation complexity. Security of a 171 - 180 bit ECC key is equivalent to RSA cryptosystem with 1024-bit key [44] which makes ECC a better choice for network communications, especially resource stringent wireless communications as well as on smartcards and embedded devices. An effective sub-index arithmetic to attack for a carefully chosen curve is yet to be identified [48].

4 QoS Aware Adaptive Security Scheme (QaASs)

Multimedia adaptation techniques take advantage of the inherent property of video data that allows to trade quality for performance. The risk level of a threat is not constant. Adaptive security takes the luxury of compromising security in order to refrain from becoming performance bottleneck for other operations. Since, a large curve offers better security than a small curve, we can assume the use of ECC schemes with curves in accordance to the risk level. Models presented in [3], [5] and [13] are promising for designing context and QoS aware adaptive security systems. Identifying measurable and adaptable security and service parameters within give constraints are key elements towards designing an adaptable system. In this work, we consider two adaptation dimensions, *security* and *multimedia*. We make an effort to identify adaptable attributes from each dimension.

The core of QaASs is an adaptation scheme that makes decisions based on a number of attributes. The scheme is tailored to adapt both cryptography and multimedia properties, given service constraints. We explain the operational procedure QaASs presenting four different service constraints. We assume that H.264/AVC is used for video encoding.

A *crypto scheme* is a unique combination of a cryptography algorithm and its properties such as internal state size, block size, number of rounds, mode of operation and key length. For example, ECC with a 128-bit curve and ECC with a 384-bit curve are two separate crypto schemes.

The receiver of real-time video provides periodic feedback to the streaming source. A feedback packet PKT_{fb} is sent by the receiver every T_{fb} interval, measured in seconds and customizable for a service session. PKT_{fb} only contains a parameter set. It would be ideal to send PKT_{fb} over a guaranteed service like TCP. Table 1 shows an example of PKT_{fb} parameter set.

Table 1. Feedback packet parameters set

Parameter	Symbol	Unit
Average transmission delay	T_{tr_d}	milliseconds
Average playback frame rate	FPS_{pb}	frames/second
Frame delay jitter	J_{f_d}	milliseconds
Decryption throughput	TP_{dec}	KBps
Frame loss percentage	F_{loss}	%(percentage)

The *crypto threshold*, $CRP_{threshold}$, indicates the maximum time allowed for a *video data unit* (slice, frame or a GOP) to spend in cryptography operations (encryption and decryption). Real-time video involves encoding and decoding. In order to cope with unexpected transmission delay and jitter, streaming applications incorporate a technique, called *jitter buffer* [22]. Moreover, introducing additional delay at the beginning of playback, to allow the decoder to receive enough frames, could help to mitigate the effect of unexpected delay and jitter. The goal is to avoid choppiness i.e., prevent the video from stalling during playback. Determining $CRP_{threshold}$, with different types of delays and other related parameters in consideration, is a problem in its own right. We consider this as a candidate for future study.

Figure 1 is a high level system diagram showing different system entities and their relations. The video traffic evaluation module evaluates received video traffic and forwards results to the feedback module. The video traffic evaluation module also decides if it should forward the received packet to the decoder or ignore it (because of delay or lost fragment). The feedback module sends periodic PKT_{fb}'s to the source. Feedback evaluator analyzes PKT_{fb}'s and results are forwarded to the adaptation module. The adaptation module decides what adaptation to be carried out and control source video traffic accordingly. The key management procedure is described in the following section.

Adaptation Option One (ADPT 1): *FPS is fixed and all video data must be encrypted.*

Since, the aim is to support a diverse range of devices, different stations have different processing capabilities. Therefore, the throughput of crypto schemes varies across stations. Each station maintains a performance profile of crypto schemes, CRP_{prf}. CRP_{prf} contains encryption and decryption throughputs for different crypto schemes, which are pre-populated. Table 2 is an example of the information stored in a CRP_{prf}. The throughput is the amount of data encrypted or decrypted by the crypto scheme per second, measured in KBps. During the service setup phase, communicating parties exchange their CRP_{prf}.

Let us assume that u is the streaming source and v is the receiver. The maximum size of a video frame is n bytes. T_{enc} is the time required to encrypt n bytes at station u. n_{enc} is the size of the encrypted payload, $n_{enc} \geq n$. T_{dec} is the time required to decrypt n_{enc} bytes at station v. Here, time is measured in milliseconds. $T_z = T_x + T_y$, is the playback time at v for a video frame that was originally available at u at time T_x. Thus, the scheduled transmission time

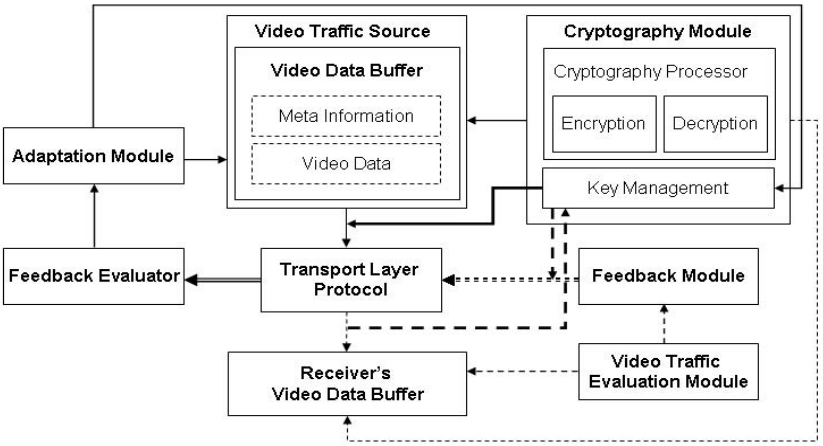


Fig. 1. High level system diagram of QaASs. The arrow-headed solid lines indicate operations at the source. The arrow-headed broken lines indicate operations at the receiver. Double lines signify PKT_{fb} communications. Thicker single lines are for key management as described for ADPT 1.

Table 2. Cryptography algorithm performance profile

Crypto Scheme ID	Algorithm	Size of Internal State or Block Size (bits)	Key Size or Curve Size (bits)	Encryption Throughput (KBps)	Decryption Throughput (KBps)
1	RC4	2064	256	993.80	996.79
4	Triple-DES	64	168	930.50	931.73
5	AES	128	128	963.57	972.95
6	AES	128	256	950.87	952.50
7	Salsa20	128	512	921.23	933.66
8	Salsa20	256	512	895.35	919.56
9	ECC	N/A	256	61.93	84.36
10	ECC	N/A	384	33.11	46.72

is $T_x + T_{enc}$. T_y is the maximum allowed playback delay for the video frame. So, we can write,

$$T_y \geq T_{enc} + T_{tr\_d} + T_{dec} \tag{1}$$

where T_{tr_d} is the transmission delay measured in milliseconds. In order to satisfy (1), we can choose a crypto scheme from the $CRP_{profile}$. We can say that T_y is a function of encoding and decoding delay, $CRP_{threshold}$, initial pause time and jitter buffer delay. For ADPT 1, $CRP_{threshold}$ indicates the maximum time allowed for a video frame to spend in cryptography operations. Hence, we must have

$$CRP_{threshold} \geq T_{enc} + T_{dec} \tag{2}$$

For each streaming session, the sender and receiver maintain the average encryption and decryption throughputs, over a predefined duration (e.g., 1 second for a 30 fps video). $T_{enc} = \frac{n_{avg}}{TP_{enc}}$, where n_{avg} is the average frame size in KB and TP_{enc} is the average encryption throughput in KBps. The T_{dec} calculation follows the same procedure and considers only the decodable frames in a received GOP. Upon receiving a PKT_{fb}, the streaming source verifies if (2) is satisfied. If not, from the recipient's $CRP_{profile}$, the source chooses a crypto scheme for which (2) is satisfied.

By switching to a higher throughput but weaker crypto scheme, to some extent, we are compromising security. Key agreement in ECIES is achieved using ECDH (Elliptic Curve Diffie-Hellman). We assume, communicating pairs exchange and agree upon ECC domain parameters, the elliptic curve E and a point P on E, during the session setup phase. They are not changed by the rekeying procedure. The source u and receiver v generate large enough random integers a and b respectively. u computes a point aP on E and sends it to v. v computes a point bP on E and sends it to u. u and v compute $a(bP)$ and $b(aP)$ respectively. The sizes of aP and bP are the same as P. The shared secret x is the x-coordinate of the point abP on E. A symmetric key can be derived from x for data encryption. We propose two rekeying options. The first option is, only u generates a new random integer a' and computes $a'P$ and sends it to v. A new symmetric key can be derived from x'. This approach would require only one-way communications. In the second option both u and v generate new random integers a' and b' respectively and therefore, a new shared secret, x'. We would require two-way communications for this option. It is possible that information exchanged for rekeying be sent separately or if possible, piggyback on data traffic. The latter reduces traffic overhead. Another important issue is the rekeying frequency. A higher frequency would require exchange of a lot more information compared to a low rekeying frequency.

Adaptation Option Two (ADPT 2): Crypto scheme and FPS are fixed. Encrypting frames containing inter coded macroblocks (e.g., P and B-frames) are optional.

The more frames are encrypted, the higher the overall delay is. Frames received out-of-order, due to delay, are discarded by the decoder and therefore, would only contribute to resources exhaustion and bandwidth wastage. In [29], we demonstrated how selective encryption influences overall transmission delay using three scenarios, encrypting all frames, only I and P-frames and only I-frames. In theory, if no I-block is present, in H.264/AVC, P and B-frames cannot be decoded without their reference I-frame. ADPT 2 takes advantage of the above property of H.264/AVC and makes encrypting inter coded frames optional. Along with I-frames, only P-frames may be chosen for encryption. Another option is not to encrypt either P or B-frames at all. Upon receiving PKT_{fb}, the streaming source verifies if (2) is satisfied. If (2) is not satisfied, selective encryption is employed. The adaptation decision is applied on a trial basis, i.e., as long as (2) is not

Table 3. H.264/AVC selective encryption options

Option	I-frame	P-frame	B-frame
i	enc	enc	enc
ii	enc	enc	x
iii	enc	x	x

satisfied. Firstly, only the B-frames are not encrypted. Secondly, P and B-frames are not encrypted. Table 3 summarizes the selective encryption options.

Adaptation Option Three (ADPT 3): Crypto scheme is fixed and all the transmitted video data must be encrypted. It is optional to transmit video frames containing inter coded macroblocks.

ADPT 3 is similar to ADPT 2 but considers a more restricted scenario. Transmission of frames containing inter coded macroblocks (e.g., P and B-frames) is optional. The adaptation decision is applied on a trial basis as in ADPT 2. Discarding non-intra coded frames results in reduced amount of encrypted data; hence, reduction in overall cryptography delay. This way, we can gain streaming performance in expense of video quality.

Adaptation Option Four (ADPT 4): Crypto scheme is fixed and all the transmitted video data must be encrypted. Variable frame rate (FPS) is allowed.

This option adapts the frame rate. A high FPS means that more frames have to be encrypted; thus, higher cryptography delay overhead. For ADPT 4, we replace TP_{dec} in PKT_{fb} by cumulative decryption time, $\sum T_{dec}$, as in (3).

$$\sum T_{dec} = \sum_{t=0.0s}^{t=1.0s} T_{dec\_frame} \tag{3}$$

Here, $\sum_{t=0.0s}^{t=1.0s} T_{dec\_frame}$ is the cumulative decryption time for frames received over a 1 s period. The source also maintains cumulative encryption time, $\sum T_{enc}$, calculated the same as above. If, $\sum_{t=0.0s}^{t=1.0s} frame\_enc$ is the total number of frames encrypted over 1 s, for a video sequence coded at 30 fps with 30 frames in each GOP, $\sum_{t=0.0s}^{t=1.0s} frame\_enc = GOP_{length}$, where GOP_{length} is the GOP length of the source video. The $\sum T_{dec}$ considers only the decodable frames in the received GOP(s). We replace $CRP_{threshold}$ in (2) by $\sum CRP_{threshold}$ where,

$$\sum CRP_{threshold} \geq \sum T_{enc} + \sum T_{dec} \tag{4}$$

In adaptation ADPT 4, we verify crypto threshold using (4). Upon receiving PKT_{fb}, in case (4) is not satisfied, the adaptation decision is applied on a trial basis. FPS is gradually decreased by a margin (e.g., 5 frames) till an acceptable minimum. Although, reduced FPS effects quality of experience, the required level of security is maintained.

5 Simulation

We used Network Simulator (NS) to verify our proposal. We follow a well accepted video traffic simulation approach. Video traffic is generated using the video traffic trace derived from the original H.264/AVC encoded video (coded using JM 1.7 [33]). The video traffic trace acts as a descriptor for the compressed video data. We used the 4:2:0 YUV *foreman* video sequence in CIF format containing 300 frames and I, P and B-slices. For the ADPT 4, we used *foreman* video coded at 30 fps, containing only I and P-frames and the GOP length is 30. The cryptography module is implemented using the Crypto++ 5.6.1 library [6] and integrated into NS version 2.28. Cryptography operations are carried out in real time on actual video data, during video traffic simulation. Simulation was carried out in a network with n^2 nodes in a $k \times k$ area, positioned in a $n \times n$ grid topology. Nodes in a grid cell are within each others communications range. The initial positions of the source and receiver are at the opposite ends of the diagonal of the $n \times n$ square grid. Table 4 lists the common network and simulation parameters. Video traffic starts at the 10^{th} second. This is to allow the proactive routing protocol some time to build the routing tables. Each slice in the H.264/AVC sequence is encrypted individually. Results are confirmed with a 95% confidence level.

Table 4. Network and simulation parameters

Network and Simulation Parameters	
Number of nodes	36
Network topology	Grid
Mobility	Random Waypoint, Velocity - 0.5 m/s
Antenna and Radio propagation model	Omni directional and Two Ray Ground Reflection
Radio range	250 m
Modulation	OFDM
MAC	802.11e EDCA
Routing protocol	MP-OLSR
Maximum data rate	2 Mbps
Basic data rate	1 Mbps
Node addressing	IPv6
Transport layer protocols	UDP for video data and TCP for others
Simulation duration	100 seconds

In order to observe the effect, the adaptation is activated at the 19^{th} second. For ADPT 1, 2, and 3, $CRP_{threshold}$ is set to 30 ms. It is considered that for real-time video, latency between two consecutive pictures should not be more than 150 ms [26]. $CRP_{threshold}$ is chosen with encoding, decoding and transmission delay in consideration. We begin simulation with encrypting video data using ECC with a 348-bit curve. We compare results of the following five different simulation scenarios, indexed S1-S5:

S1. Without adaptation.
S2. Single video traffic flow with adaptation and no rekeying.
S3. In presence of six video traffic flows with adaptation and no rekeying.
S4. Single video traffic flow with adaptation and with rekeying option one (frequency: once every 1 s).
S5. Single video traffic flow with adaptation and with rekeying option two (frequency: once every 1 s).

Figures 2, 4 and 5 show average cryptography delay and transfer time for ADPT 1, 2 and 3 respectively. For S2, gain in transfer time over absence of adaptation are 9, 12 and 24 s for ADPT 1, 2 and 3 respectively. Using ADPT 1, in S2, in order to satisfy $CRP_{threshold}$, crypto scheme is changed from ECC with a 384-bit curve to ECC with a 192-bit curve. In S3, five cryptography processes cause noticeable delay for each flow, approximately an average 74 ms (Fig. 2). As a result of which the crypto scheme is changed from ECC with a 384-bit curve to ECC with a 128-bit curve. After withdrawing four out of five flows, the crypto scheme is moved up to ECC with a 192-bit curve. Since, the rekeying option two requires two-way communications and in-session key agreement, the delay overhead in S5 is slightly higher than S4. Figure 3 shows a comparison of network traffic caused by S1, S2, S4 and S5. S5, with frequency once in every 0.5 s, generates highest amount of bytes among the ones with adaptation, which is still half the amount compared to S1. For evaluation of ADPT 4, $CRP_{threshold}$ is set to 1000 ms and initially there are five video traffic flows. For the selected simulation parameters, ultimately the frame rate is reduced to just 5 fps. After withdrawing four out of five flows, the frame rate is moved up to 15 fps and a 11 seconds gain in playback time.

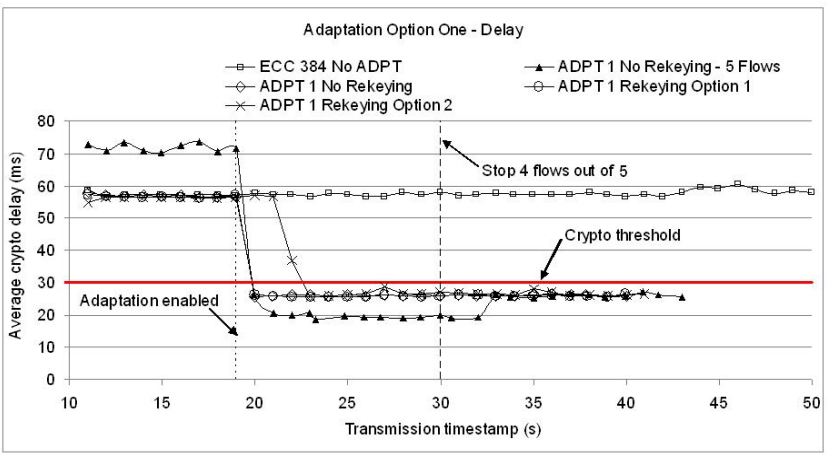


Fig. 2. Adaptation option one for different scenarios

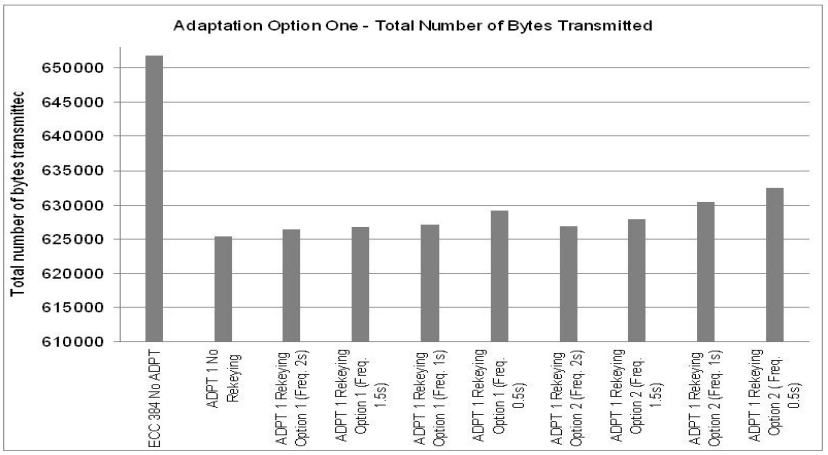


Fig. 3. Adaptation option one with rekeying

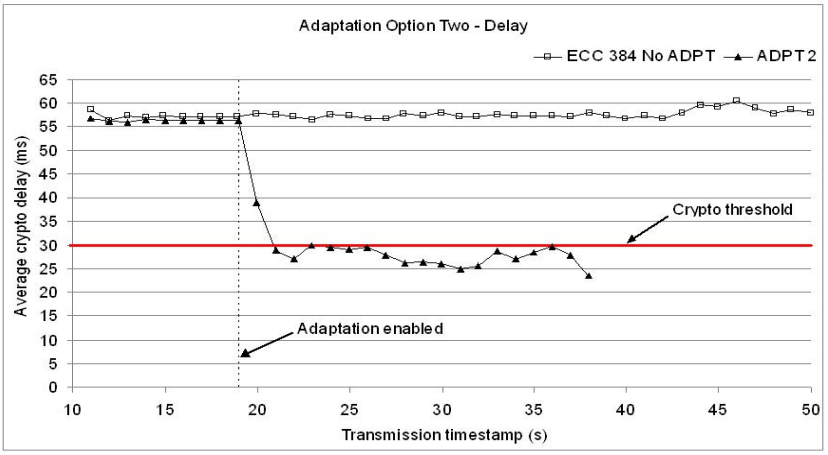


Fig. 4. Comparison of delay: no adaptation vs adaptation option two

In ADPT 1, in order to meet $CRP_{threshold}$, we switch to a higher throughput ECC crypto scheme with a smaller curve and therefore, security is being compromised while original video quality is maintained. In ADPT 2, based on receiver's feedback, we decide if frames containing inter coded macroblocks (e.g., P and B-frames) should be encrypted. When P or B-frames are not encrypted, we are transmitting unencrypted data. Therefore, security is compromised but original video quality is maintained. According to Agi and Gong [1], the presence of I-blocks in unencrypted P and B-frames is a security hole. A series of P and B-frames could carry enough information if their base frames are correlated. A frame containing an unencrypted I-block, being referenced by blocks in

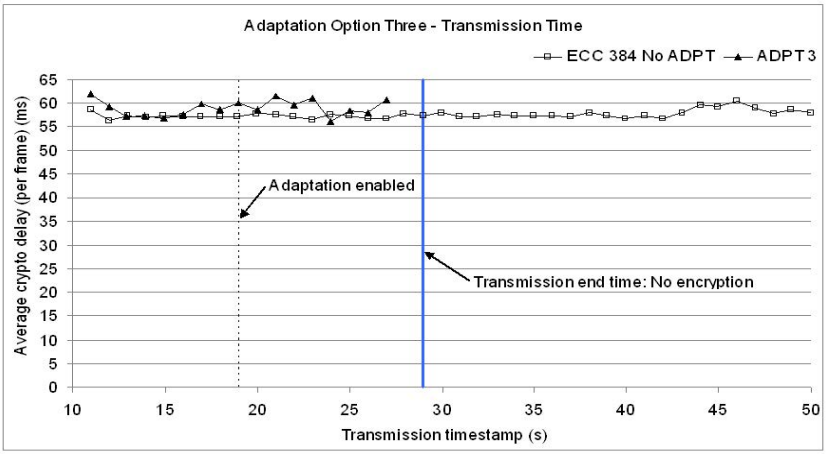


Fig. 5. Comparison of transmission time: no adaptation vs adaptation option three

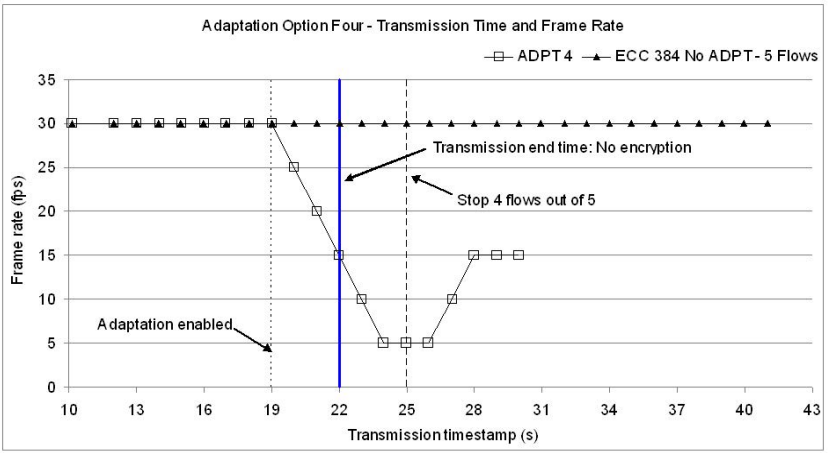


Fig. 6. Comparison of transmission time: no adaptation vs adaptation option four

subsequent frames, can be decoded. Therefore, in the case of both ADPT 1 and 2, with sufficient computing resources, it might be possible for a crypto analyst to glean information about the video data. In ADPT 3, to satisfy $CRP_{threshold}$, we may not transmit the unencrypted P and B-frames. Hence, video quality is compromised but security is maintained. In ADPT 4, frame rate is adapted in order to meet $CRP_{threshold}$. Reducing the frame rate decreases the video playback quality. Since all video data are encrypted with the original crypto scheme, security is maintained. Table 5 summarizes the key elements of comparison of the four adaptation options.

Table 5. Summary of comparison of the four adaptation options

	Adaptation Option One	Adaptation Option Two	Adaptation Option Three	Adaptation Option Four
Adapted Property	Crypto Scheme	Inter Coded Frame Encryption	Inter Coded Frame Transmission	Video FPS
Possibility of Compromising Security	Yes	Yes	No	No
Possibility of Compromising Video Quality	No	No	Yes	Yes

6 Conclusion

The goal of this work was to develop a solution that addresses the issue of delay overhead caused by cryptography operations. We perceived the addressed problem in the context of real-time multimedia streaming in MANETs. We have proposed a runtime adaptation mechanism that adapts cryptography and/or multimedia service properties in order to meet desired QoS while maintaining the required level of security. The adaptation mechanism utilizes service feedback from the receiver in real-time. We have presented four different adaptation options exemplifying different application requirements. Our evaluation shows that, in presence of adaptation, video transfer time is reduced by a significant margin while using computationally intensive ECC crypto schemes. We have also shown that, while choosing an ECC crypto scheme with higher throughput and lower security reduces transmission delay, periodic rekeying is a viable option to elevate security with less than apprehensible effect on performance. Adaptation options three and four, though reduce quality, improves video transfer time without compromising security. We have also presented a taxonomy of the presented adaptation options.

In our work, we have only considered H.264/AVC. The concept can be extended for H.264/SVC and 3D/Stereoscopic coding [24]. In addition to unicasting, we can verify and adapt the technique for multicasting. For our evaluation, we have assumed the value of crypto threshold (e.g., 30 ms). Ideally, the value of the crypto threshold should depend on the network and application. An algorithm to determine the value of crypto threshold based on network, application, user and security contexts can be a candidate for future work. We also think Multi Criteria Decision Making [4] approaches such as AHP based solution can be developed to realize multidimensional (e.g., adapting security and QoS parameters simultaneously) adaptation decisions.

References

[1] Agi, I., Gong, L.: An Empirical Study of Secure MPEG Video Transmissions. In: Proceedings of the Symposium on Network and Distributed System Security (SNDSS), pp. 137–144. IEEE Computer Society, Washington, DC (1996)

[2] Alampalayam, S., Kumar, A.: An adaptive security model for mobile agents in wireless networks. In: IEEE Global Telecommunications Conference (GLOBECOM), vol. 3, pp. 1516–1521 (December 2003)

[3] Alia, M., Lacoste, M., He, R., Eliassen, F.: Putting together QoS and security in autonomic pervasive systems. In: Proceedings of the 6th ACM Workshop on QoS and Security for Wireless and Mobile Networks (Q2SWinet), pp. 19–28. ACM, New York (2010)

[4] Ben Mahmoud, M., Larrieu, N., Pirovano, A., Varet, A.: An adaptive security architecture for future aircraft communications. In: IEEE/AIAA 29th Digital Avionics Systems Conference (DASC), pp. 3.E.2-1–3.E.2-16 (October 2010)

[5] Blasi, L., Savola, R., Abie, H., Rotondi, D.: Applicability of security metrics for adaptive security management in a universal banking hub system. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (ECSA), pp. 197–204. ACM, New York (2010)

[6] Crypto++, `http://www.cryptopp.com`

[7] Devore, J.: Probability and Statistics for Engineering and the Sciences. Cengage Learning (2011)

[8] Djenouri, D., Khelladi, L., Badache, A.: A survey of security issues in mobile ad hoc and sensor networks. IEEE Communications Surveys Tutorials 7(4), 2–28 (2005)

[9] Dong, H., Gibson, J., Kokes, M.: SNR and bandwidth scalable speech coding. In: IEEE International Symposium on Circuits and Systems (ISCAS), vol. 2, pp. II-859–II-862 (2002)

[10] Furht, B.: Encyclopedia of Multimedia. Springer Reference. Springer (2008)

[11] Gibson, J., Servetti, A., Dong, H., Gersho, A., Lookabaugh, T., De Martin, J.: Selective encryption and scalable speech coding for voice communications over multi-hop wireless links. In: IEEE Military Communications Conference (MILCOM), vol. 2, pp. 792–798 (October-November 2004)

[12] ITU-T Rec. H.264 and ISO/IEC 14496-10:2005 (E) (MPEG-4 AVC). H.264: Advanced video coding for generic audiovisual services. Technical report (2005)

[13] He, R., Lacoste, M., Pulou, J., Leneutre, J.: A DSL for Specifying Autonomic Security Management Strategies. In: Garcia-Alfaro, J., Navarro-Arribas, G., Cavalli, A., Leneutre, J. (eds.) DPM 2010 and SETOP 2010. LNCS, vol. 6514, pp. 216–230. Springer, Heidelberg (2011)

[14] IEEE. Part 11 Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications: Medium Access Control (MAC) Quality of Service (QoS) Enhancements. IEEE Std P802.11e/D13.0 (2005)

[15] Iqbal, R., Shahabuddin, S., Shirmohammadi, S.: Compressed-domain spatial adaptation resilient perceptual encryption of live H.264 video. In: 10th International Conference on Information Sciences Signal Processing and their Applications (ISSPA), pp. 472–475 (May 2010)

[16] ISO/IEC. ISO/IEC 21000-7:2007 - Information technology - Multimedia framework (MPEG-21) - Part 7: Digital Item Adaptation. Technical report, International Organization for Standardization (2007)

[17] ITU-T. 5-, 4-, 3- and 2-Bits Sample Embedded Adaptive Differential Pulse Code Modulation (ADPCM) (1990)
[18] Kamphenkel, K., Blank, M., Bauer, J., Carle, G.: Adaptive encryption for the realization of real-time transmission of sensitive medical video streams. In: International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), pp. 1–6 (June 2008)
[19] Macq, B., Quisquater, J.: Cryptology for digital TV broadcasting. Proceedings of the IEEE 83(6), 944–957 (1995)
[20] Meyer, J., Gadegast, F.: Security mechanisms for Multimedia data with the Example MPEG-1 video, Project description of SECMPEG (2000)
[21] Nijim, M., Ali, A.: AdSeD: An Adaptive Quality of Security Control in Disk Systems. In: 11th IEEE Intl. Conf. on Computational Sci. and Eng. (CSE), pp. 421–428 (July 2008)
[22] Oklander, B., Sidi, M.: Jitter Buffer Analysis. In: Proceedings of the 17th International Conference on Computer Communications and Networks (ICCCN), pp. 1–6 (August 2008)
[23] Oliveira, T., Oliveira, S., Macedo, D., Nogueira, J.: An adaptive security management model for emergency networks. In: 7th Latin American Network Operations and Management Symposium (LANOMS), pp. 1–4 (October 2011)
[24] Onural, L.: An Overview of Research in 3DTV. In: 14th International Workshop on Systems, Signals and Image Processing (IWSSIP), p. 3 (June 2007)
[25] Pereira, R., Tarouco, L.: Adaptive Multiplexing Based on E-model for Reducing Network Overhead in Voice over IP Security Ensuring Conversation Quality. In: 4th Intl. Conf. on Digital Telecommunications (ICDT), pp. 53–58 (July 2009)
[26] Perkins, C.: RTP: audio and video for the internet. Kaleidoscope Series. Addison-Wesley (2003)
[27] Perkins, D., Hughes, H.: A survey on quality-of-service support for mobile ad hoc networks. Wireless Communications and Mobile Computing 2(5), 503–513 (2002)
[28] Preda, S., Cuppens, F., Cuppens-Boulahia, N., Garcia-Alfaro, J., Toutain, L.: Dynamic deployment of context-aware access control policies for constrained security devices. J. Syst. Softw. 84(7), 1144–1159 (2011)
[29] Reza, T.: QoS Aware Adaptive Security Scheme for Video Streaming in MANETs. Master's thesis, School of Computer Science, Carleton University, Ottawa, Ontario, Canada (2012), `http://people.scs.carleton.ca/~barbeau/Theses/Tashin_Reza.pdf`
[30] Samad, F., Makram, S.: Adaptive security established on the requirements and resource abilities of network nodes. In: IEEE 35th Conference on Local Computer Networks (LCN), pp. 752–755 (October 2010)
[31] Schwarz, H., Marpe, D., Wiegand, T.: Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. IEEE Transactions on Circuits and Systems for Video Technology 17(9), 1103–1120 (2007)
[32] Shnitko, A.: Adaptive security in complex information systems. In: Proc. of the 7th Korea-Russia Intl Symp. on Sci. and Tech. (KORUS), vol. 2, pp. 206–210 (July 2003)
[33] Sühring, K.: H.264/AVC JM Reference Software, `http://iphome.hhi.de/suehring/tml/download/`
[34] Son, H., David, R., Thuraisingham, B.: Improving timeliness in real-time secure database systems. SIGMOD Rec. 25(1), 29–33 (1996)
[35] Spanos, G., Maples, T.: Performance Study of a Selective Encryption Scheme for the Security of Networked, Real-Time Video. In: Fourth International Conference on Computer Communications and Networks, pp. xviii+683 (September 1995)

[36] Stallings, W.: Cryptography and network security: principles and practice. Prentice Hall (2011)
[37] Taddeo, A., Marcon, P., Ferrante, A.: Negotiation of security services: a multi-criteria decision approach. In: Proceedings of the 4th Workshop on Embedded Systems Security (WESS), pp. 4:1–4:9. ACM, New York (2009)
[38] Tang, L.: Methods for encrypting and decrypting MPEG video data efficiently. In: Proceedings of the Fourth ACM International Conference on Multimedia (MULTIMEDIA), pp. 219–229. ACM, New York (1996)
[39] Vaidya, B., Denko, M., Rodrigues, J.: Secure Framework for Voice Transmission over Multipath Wireless Ad-Hoc Network. In: IEEE Global Telecommunications Conference (GLOBECOM), pp. 1–6 (December 2009)
[40] Van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. SIGPLAN Not. 35(6), 26–36 (2000)
[41] Van Deursen, D., Van Lancker, W., Van de Walle, R.: On media delivery protocols in the Web. In: IEEE Intl. Conf. on Multimedia and Expo (ICME), pp. 1028–1033 (July 2010)
[42] Venkatramani, C., Westerink, P., Verscheure, O., Frossard, P.: Securing media for adaptive streaming. In: Proceedings of the Eleventh ACM International Conference on Multimedia (MULTIMEDIA), pp. 307–310. ACM, New York (2003)
[43] Wiegand, T., Sullivan, G., Bjontegaard, G., Luthra, A.: Overview of the H.264/AVC video coding standard. IEEE Transactions on Circuits and Systems for Video Technology 13(7), 560–576 (2003)
[44] Wiener, M.: Performance Comparison of Public-key Cryptosystems (1998), `http://www.rsa.com/rsalabs/pubs/cryptobytes.html`
[45] Yau, S., Yin, Y., An, H.: An Adaptive Tradeoff Model for Service Performance and Security in Service-Based Systems. In: Proceedings of the 2009 IEEE International Conference on Web Services (ICWS), pp. 287–294. IEEE Computer Society, Washington, DC (2009)
[46] Yi, J., Adnane, A., David, S., Parrein, B.: Multipath optimized link state routing for mobile ad hoc networks. Ad Hoc Networks 9, 28–47 (2011)
[47] Younis, M., Krajewski, N., Farrag, O.: Adaptive security provision for increased energy efficiency in Wireless Sensor Networks. In: IEEE 34th Conf. on Local Comp. Networks (LCN), pp. 999–1005 (October 2009)
[48] Yu, B.: Establishement of elliptic curve cryptosystem. In: IEEE International Conference on Information Theory and Information Security (ICITIS), pp. 1165–1167 (December 2010)
[49] Zou, J., Lu, K., Jin, Z.: Architecture and fuzzy adaptive security algorithm in intelligent firewall. In: Proceedings of MILCOM (MILCOM), vol. 2, pp. 1145–1149 (October 2002)

A Case Study of Side-Channel Analysis Using Decoupling Capacitor Power Measurement with the OpenADC

Colin O'Flynn and Zhizhang Chen

Dalhousie University, Halifax, Canada
{coflynn,z.chen}@dal.ca

Abstract. When capturing power measurements for processing with side-channel analysis, there are many options with regards to both how the measurement is taken, and also how that measurement is digitized. This work concentrates on a new technique which measures the current through a decoupling capacitor, with a probe that can easily be built in any electronics lab. In addition an open-source digitizer board is presented, which is specifically designed to measure the signals required for side-channel analysis. The techniques presented in this work facilitate sharing of repeatable measurement techniques: the measurement environment presented can easily be duplicated at a very low cost.

Keywords: side-channel analysis, decoupling, acquisition, case study.

1 Introduction

Using the power consumption of a device as a 'side channel' to derive secrets held inside the device was first presented in 1998[1]. The initial two attacks, called Differential Power Analysis (DPA) and Simple Power Analysis (SPA) have since been augmented by even more powerful attacks such as template attacks[2] or Correlation Power Analysis (CPA)[3]. This work does not concentrate on the attacks; instead, this work focuses on how the power consumption of a device under attack is measured. First, an overview of current technologies used in the capture of power traces will be presented. From this we can define the requirements for a capture system, before moving onto an implementation of a capture system meeting these requirements. In addition to a low-cost capture system, a simplified probe type is proposed, which has the advantage of being easily reproducible by other researchers. Finally a comparison of the proposed capture architecture and probe will be compared to commercially available solutions, as typically used in recent literature.

2 Review of Capture Techniques

2.1 Probe Type

There are two general classes of probes used for measuring power consumption: a resistive shunt as used in the original work [1], or an electromagnetic (EM)

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 341–356, 2013.

probe[4]. EM probes have been shown to result in more successful attacks[5], with the advantage that EM probes do not require any modification to the device under attack, and can even be performed at a moderate range[6].

Many types of EM probes have been used in published work, including commercially manufactured probes. Comparison of different probe constructions is found in [7, 8]. Smaller probes can be scanned over the chip surface to pick out specific features, such as bus/data lines.

2.2 Acquisition

The required acquisition characteristics depend on both the target under attack, and the type of attack being carried out. Considerations with regards to the target under attack include the clock frequency, whether the cryptographic algorithm is in hardware (HW) or software (SW), technology used for the chip, and whether or not countermeasures have been implemented. Typically the capture oscilloscope achieves around 1 GS/s, as shown in Table 1.

Table 1. A few examples of capture rates in recently published papers. Sample rates only appear if the tested attack was successful at that sample rate.

Reference	Sample Rate(s) - MS/s	Target Type
[9]	5000	HW - 24 MHz
[7]	500, 2000	HW
[10]	200	SW
[8]	125, 250, 500	SW - 24 MHz
[11]	500, 1000	HW

3 Ideal Acquisition System - Requirements

3.1 External Clock Inputs

Commercial oscilloscopes typically provide their own sampling clock which is not synchronized to the device clock. In many devices, however, the device clock is readily available either as a digital signal or by adding a buffer circuit to the crystal oscillator. The sample clock can be derived from the device clock to measure a consistent point; for example it can be used to measure the power consumption on the clock edge. A comparison of measurements taken with an unsynchronized and synchronized sample clock is shown in Fig. 1. In section 6 it will be demonstrated that this synchronized sample clock significantly relaxes the requirement of using a high sample rate for certain attacks.

Note that sample clock synchronization is different from the trigger input that all oscilloscopes provide. With a real-time oscilloscope, the internal sample clock of the oscilloscope will be running at all times, and the sample occurs at the next clock edge after the trigger. Thus even though the oscilloscope is triggered at a repeatable time, there will be some random jitter between when the first

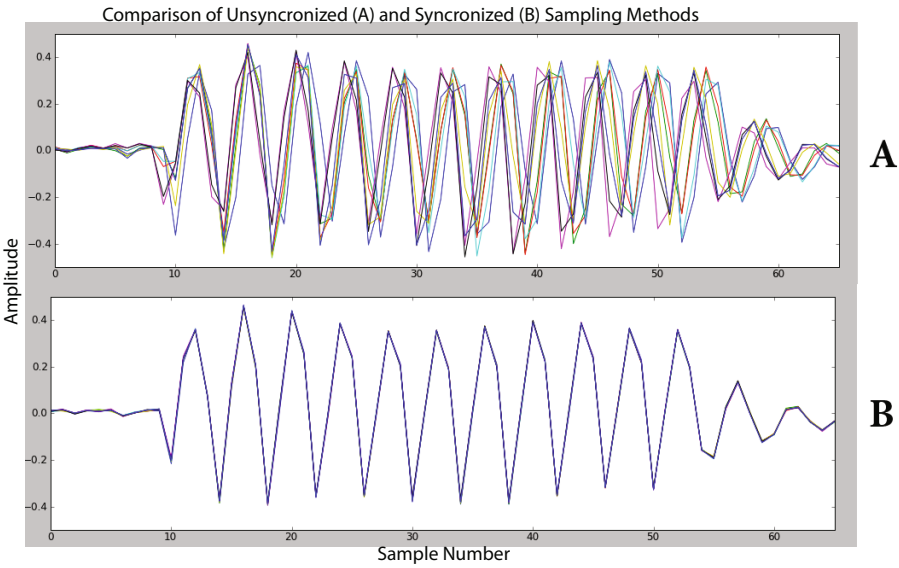


Fig. 1. Eight power samples with the same input are taken and overlaid to show consistency of measurements. In *A* the sample clock is 100 MHz but not synchronized to the device clock, whereas in *B* the sample clock is 96 MHz, but synchronized with the device clock.

sample occurs relative to this trigger for unsynchronized (free-running) sample clocks[12]. Some oscilloscopes do provide a synchronous sampling ability, such as the CleverScope with the 'external sampler clock input', or the PicoScope 6000 series.

If the clock frequency varies due to either countermeasures or a low-cost oscillator, this would not affect the acquisition quality, since samples are always based on the device clock.

3.2 External Clock Phase Adjust

The processing of the external clock input, the ADC, and the analog front-end will add some delay between when the rising clock occurs on the target device, and when the actual sample is recorded. In addition the point of interest for the power analysis may not lie directly on the rising edge, but sometime after this clock edge. For this reason the capture board must be able to add an adjustable delay (phase shift) between the input clock and the actual sample point.

3.3 Adjustable Gain

The output of a probe will vary with both the probe type and the circuit under analysis. For this reason, an adjustable gain amplifier is useful to amplify the

signal up to the range of the input of the digitizer. Oscilloscopes for example provide a selectable input range - this is still insufficient for H-Field probes, which require an external Low Noise Amplifier (LNA).

4 Low-Cost Acquisition Architecture

The architecture of the analog front-end which is used here is shown in Fig. 2. The features previously identified as important for side-channel analysis are included: an external clock input with adjustable phase, an internal clock, adjustable gain, and a computer interface.

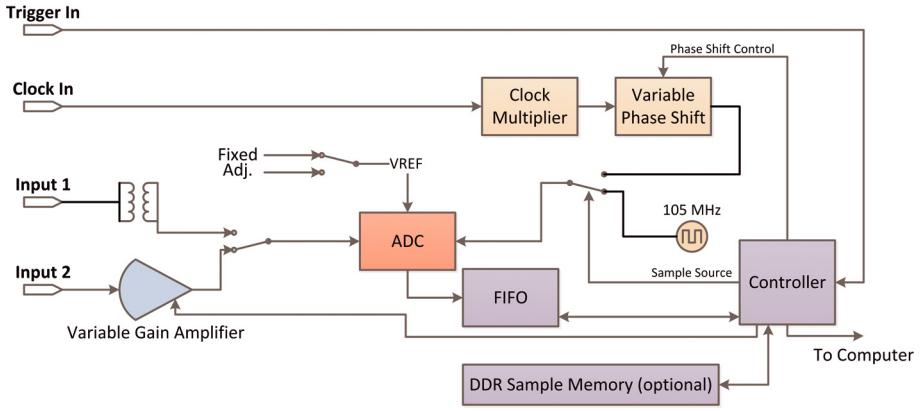


Fig. 2. Architecture of analog acquisition unit which is implemented in a combination open-source ADC board and COTS FPGA board

The analog front-end and ADC board has been released in an open-source design called the OpenADC. The OpenADC hardware consists only of the low noise amplifier (LNA), ADC, input connectors, and associated support circuitry such as power supplies. This board can be connected to most FPGA development boards with sufficient IO available - it is shown mounted on a low-cost Xilinx Spartan 6 development board from Avnet in Fig. 3. The open-source solution includes not only the PCB designs, but example FPGA source code and capture applications on the PC at [13]. The total cost of this acquisition solution is $140 US.

While the sample rate is limited by the 10-bit ADC selected to 105 MS/s, the analog bandwidth is higher to maintain information on the clock edges. When the LNA input is selected the analog bandwidth is around 110 MHz, and when the transformer-coupled input is selected the analog bandwidth is around 500 MHz. The LNA has an adjustable gain in 100 steps up to 55 dB, allowing for the direct connection of a wide range of measurement probes, including both current shunt and EM.

In addition to the analog hardware and FPGA source, the PC-based capture application source is provided. This source is written entirely in Python, providing an excellent cross-platform tool which can easily be expanded. The basic

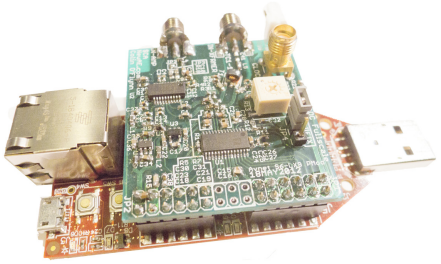


Fig. 3. The OpenADC mounted on a commercial FPGA development board. The FPGA board provides control, USB interface, and a 48M sample memory.

library provides the hardware interface code along with the graphical display. This can be easily integrated into other applications: Fig. 4 shows two example applications that make use of this library.

5 Decoupling Capacitor Power Measurement

A decoupling capacitor is designed to provide a low-impedance path for high frequency current, as typically drawn at the clock edge[14]. For side-channel analysis with a resistive shunt, the decoupling capacitor significantly worsens the measured signal [15]. The higher-frequency components, which are of interest for SCA, are flowing through the decoupling capacitor and not the shunt.

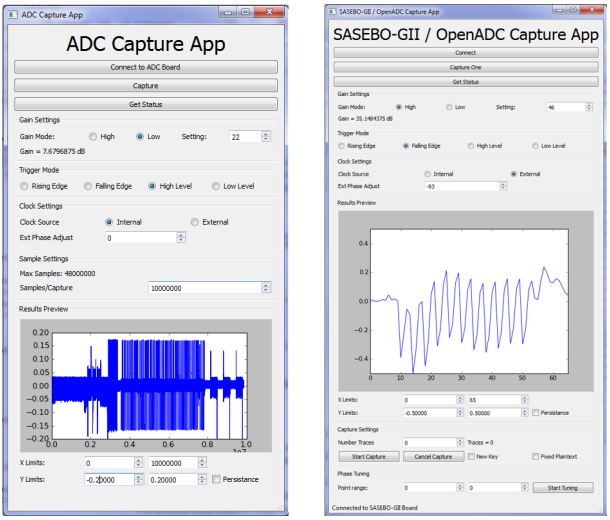


Fig. 4. Example capture applications provided. The *left* example controls only the OpenADC, and this example is a long (10E6 point) capture of a KeeLoq algorithm. The example on the *right* interfaces to the OpenADC and SASEBO-GII board to capture many traces with different plaintext data.

Measuring the current through a decoupling capacitor for side-channel analysis was first explored in [16], which used a current transformer to measure the current flowing through individual decoupling capacitors. Current transformers use the principle of induction, which dates back to Faraday's discovery in 1831[17], to measure current flowing in a conductor without the necessity of breaking the conductor. Using induction to measure current through a decoupling capacitor in-place has also been demonstrated, but such papers employed the measurements for the design of power distribution systems, and not for side-channel analysis [18–20]. This paper builds on such previous work by looking at the performance of the inductive pickup for side-channel attacks, and the physical considerations for its use.

The method thus proposed is to wrap the target decoupling capacitor in a thin magnet wire, and connect this to the acquisition oscilloscope. Physically, this proposed method requires no modifications to the device under test. The localized nature of the measurement provides excellent rejection of interference, and the performance when used in side-channel attacks will be demonstrated to be slightly superior to other common methods.

6 SASEBO-GII Correlation Power Analysis (CPA) Results

The Side-channel Attack Standard Evaluation Board (SASEBO) version GII from the National Institute of Advanced Industrial Science and Technology (AIST) in Japan provides a useful reference platform for performing side-channel analysis attacks. Characterizations are available in literature of the performance of this board under various attacks[15, 21]. The attack used here is a simple Correlation Power Attack, for which the reference code is available from AIST[22], with the cryptographic core under attack being the AES core provided for the DPA Contest Version 3[22].

The performance analysis here consists of the number of traces required for the global success rate (GSR) to stay above 80%. This performance analysis was chosen to match recent publications of a similar nature [9, 23].

6.1 Measurement Setup

The measurement equipment consists of an Agilent 54831B Infiniium DSO as a reference, and the OpenADC platform presented earlier as a demonstration of low-cost capture hardware. The acquisition from the Agilent 54831B is done with code from AIST[22] which has been modified to support the scope being used, with a sampling rate at 2 GS/s. This scope does not support an external clock input. Vertical voltage scale differs depending on the measurement setup being used. For the OpenADC the sampling clock (96 MHz) is 4x the AES Core Clock (24 MHz), which is derived from the actual AES Core Clock. The OpenADC capture software is written in Python and the source code is available from [13].

Fig. 5. Shielded magnetic field probe, before wrapping in an insulator to allow safe probing of any area of the device under test

In all cases the internal voltage (VINT) of the FPGA is adjusted to 1.000 volts; this avoids any unintentional results occurring because the insertion of the current shunt will naturally reduce the voltage seen by the FPGA. The SASEBO-GII is equipped with a small adjustment range on the VINT voltage to null out the current shunt loss.

The board as shipped did not have decoupling capacitors mounted on VINT, which correspond to C46 - C52. Where a decoupling capacitor is mounted in these tests, only a single 100 nF capacitor is mounted on C46, for which a Murata GRM155R61A104KA01D size 0402 capacitor is used.

Current Shunt. The SASEBO-GII board provides connections for measuring current used by the cryptographic FPGA via a 1-ohm current shunt. This measurement uses the 'VINT' supply for the FPGA, which is measured at J2. This measurement is performed both with C46 mounted and unmounted.

H-Field Probe. An H-Field probe was constructed from a loop of semi-rigid coax. When using the 54831B oscilloscope, a MiniCircuits ZFL-1000LN Low Noise Amplifier (LNA) boosts the signal to achieve a better response. The OpenADC is directly connected to the H-Field probe, as the OpenADC contains an integrated LNA. A photo of the magnetic field probe is shown in Fig. 5. Detailed information about the construction process is found in [14], with some additional examples for side-channel analysis in [7].

Shunt Measurement on Individual Capacitors. The current through an individual capacitor was measured with a 0.22 ohm current shunt placed in series with the capacitor. The voltage was read directly from the current shunt and fed into the oscilloscope.

Power Pin Measurements. If the decoupling capacitors are not mounted, the device will naturally see drops in its voltage supply as measured at the power pin, since the power distribution system is unable to provide a low-impedance source close to the power pin. For the SASEBO-GII board, the measurement is taken on the underside of the board, on the positive pad of the decoupling capacitor specified. Each decoupling capacitor pad aligns directly with the power pin of the cryptographic FPGA, see Fig. 6

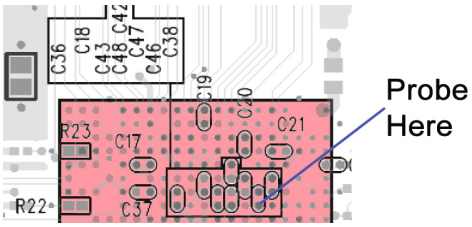


Fig. 6. The decoupling capacitors line up directly with the power pins; if the capacitors are not mounted this provides a good source to measure the ripple on the voltage rail due to high-frequency power demands. The pink square is the location of the chip under attack on the top side of the board.

Inductive Wrapping. The proposed inductive wrap method uses 7 wraps of AWG34 magnet wire around the decoupling capacitor C46. One end of the magnet wire is soldered to the negative pad of the capacitor. The other side of the wire connects through a low-noise amplifier (ZFL-1000LN) for the DSO, or directly to the OpenADC. Fig. 7 shows a detailed photo of this setup.

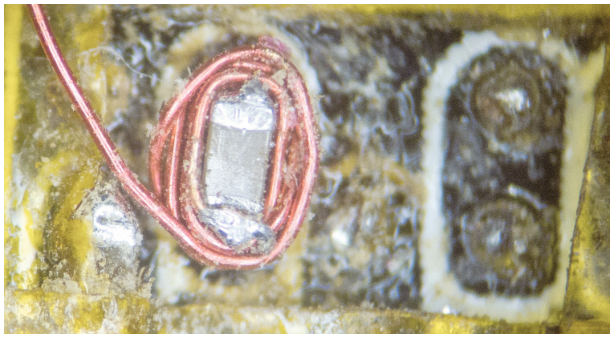


Fig. 7. 7 wraps of AWG34 magnet wire around a 0402 capacitor. The yellow visible around the capacitor is Kapton tape used to isolate the rest of the PCB.

6.2 Measurement Results

Results for the Global Success Rate (GSR) of the CPA attack are shown in Fig. 8; Table 2 provides the number of traces require for the GSR to exceed 0.8. All of these measurements are taken with the Agilent DSO, a comparison between the DSO and OpenADC platform is given in Fig. 10.

Current Shunt, H-Field Probe. In order to confirm the test setup, several of the results duplicated work done elsewhere. For example, the shunt measurement on the entire VCC-INT power system was expected to perform poorly when the single decoupling capacitor was mounted. This can be seen by comparing Fig. 2-A to Fig. 2-B. In addition, the H-Field probe should provide better results than

Table 2. Traces required to achieve 1st order Global Success Rate (GSR) higher than 80% with a Correlation Power Analaysis (CPA) attack for several measurement techniques

Measurement Method	Traces for GSR > 0.8
VCC-INT Shunt Measurement	4800
VCC-INT Shunt Measurement w/ decoupling	>5000
Inductive Pickup w/ decoupling w/ amplifier	3450
H-Field Probe w/ decoupling w/ amplifier	3850
Decoupling capacitor shunt w/ decoupling	4350
Voltage Probe	4550

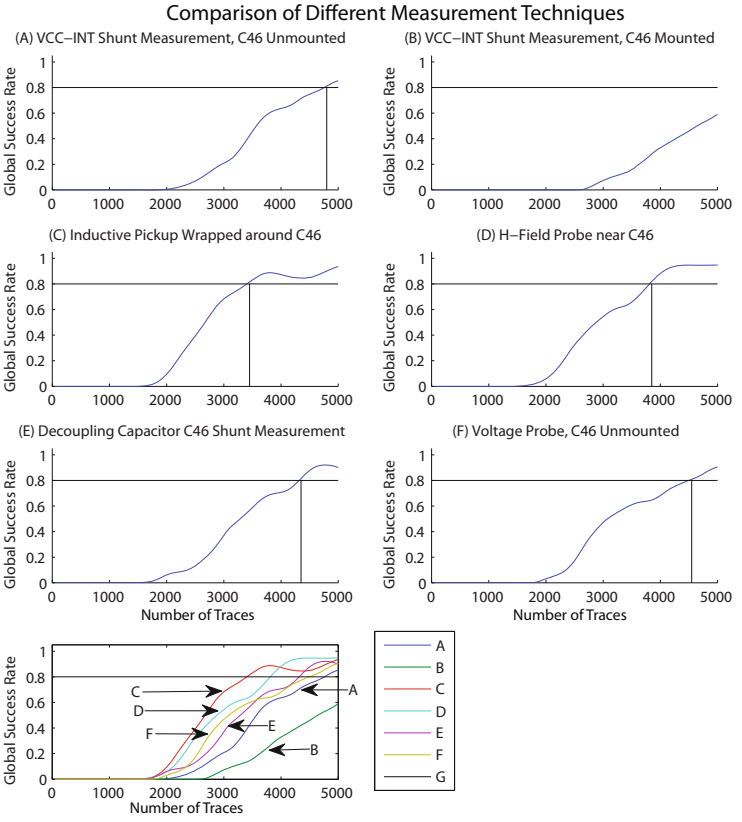


Fig. 8. The first order global success rate (GSR) vs. the number of traces processed for a simple CPA attack. *A* through *F* show different measurement techniques; the final figure shows a comparison of the first-order GSR for each of the measurement techniques. The vertical lines show the intercept of the 1st order GSR exceeding 0.8, where the numeric value of these intercepts is given in Table 2.

the shunt measurement in order to agree with [5]. This is confirmed by looking at Fig. 2-D.

Inductive Wrapping. It can be seen that the proposed measurement technique requires the smallest number of traces to achieve a GSR higher than 80% (>0.8). The signal from this technique is considerably stronger than with the H-field probe. The measured signal from the inductive wrap technique is about 10x larger in amplitude (Vp-p) than that from the H-field probe.

The stronger signal slightly relaxes the requirements of the amplifier, and means that the resulting SNR will be better compared to the H-field probe. The results here show slightly better performance for the inductive wrapping technique compared to the H-Field probe due to this improved SNR. The number of wraps used does appear to impact the GSR, as shown in Fig. 9. Here 7 wraps results in a better GSR than 2 wraps - the 7 wraps again resulted in a stronger signal, reducing noise in the measurement front-end.

Shunt on Decoupling Capacitor. The results here confirm the decoupling capacitor measurement does provide a significant improvement over attempting to measure the current drawn through the entire system. The performance is still lower than electromagnetic techniques; it is assumed that adding the shunt reduces the impedance of the capacitor, thus reducing the current which flows through it. In [16] a Current Transformer (CT) is used instead of a resistive shunt. Inserting the CT would also slightly increase the impedance, since the CT must be clamped around a wire in series with the decoupling capacitor.

Voltage Probe. The voltage probe is an extremely simple method of measuring local variations in the current demand. It does require the decoupling capacitor to be removed: for the best signal it would likely demand all nearby capacitors to be removed, as the nearby capacitors provide some additional decoupling that dampens the signal. For devices under attack which require the decoupling capacitors to run, this method may not be possible.

6.3 OpenADC Measurement Results

The results in Fig. 10 show that the OpenADC performs well using the inductive wrapping technique. The OpenADC is only sampling at 96 MSPS - but the sampling clock is synchronized to the device clock. When the sampling clock is not synchronized, it fails to recover the encryption key (GSR = 0). This agrees with previously published results on a similar board, which showed a failure of the attack at 100 MS/s [11]. The reference measurement at 2 GS/s is using the oscilloscope's internal timebase; that is to say a timebase that is unsynchronized to the device clock.

The OpenADC has fine granularity on the gain of the input signal, along with the full-scale reference voltage for the ADC. The DSO by comparison does not provide such fine granularity on the input scaling. For the inductive wrap technique it is expected that this partially contributes to the slightly better performance of

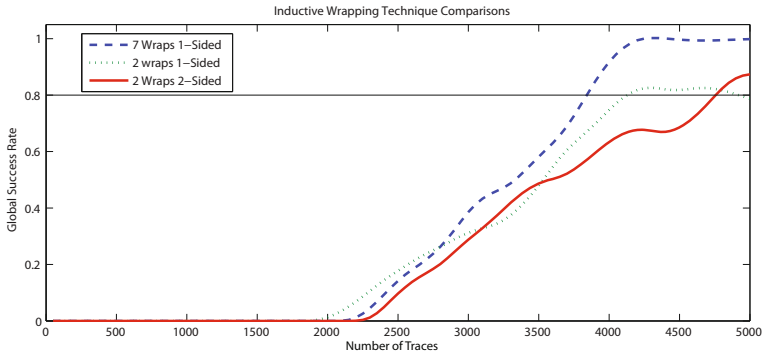


Fig. 9. Comparison of different variations of the inductive wrapping technique. The maximum number of wraps was set based on the physical ability to keep the wraps around the decoupling capacitor. An '1-sided' wrap has one end soldered to the ground pad of the capacitor as in Fig. 7, where a '2-sided' wrap has both ends of the wrapping wire connected to the oscilloscope as in Fig. 13. Appendix B provides some information about the '1-sided'/'2-sided' wrapping.

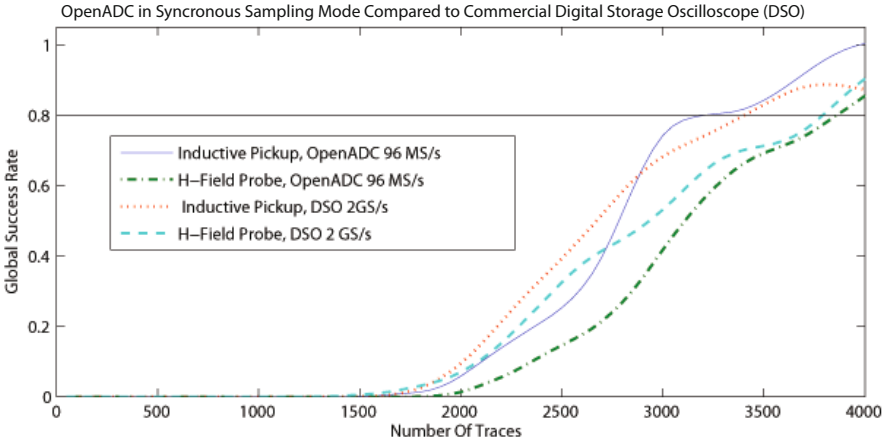


Fig. 10. Comparison of GSR for traces gathered with the OpenADC and a normal Digital Storage Oscilloscope (DSO), for both H-Field probe and inductive wrapping

the OpenADC: the number of bits used to represent the full-scale signal is higher with the OpenADC compared to the DSO, since the OpenADC allows adjustment of the signal to reach closer to the full-scale range of the ADC input.

7 Conclusions

Previous work in side-channel analysis has shown that the electromagnetic (EM) field provides better results than a current shunt. The EM probe, however, suffers

from being more complex to use: it requires careful positioning if repeatable experiments are necessary, requires a Low Noise Amplifier, and often requires an expensive high-speed oscilloscope.

This work has shown a low-cost alternative that solves both problems: rather than using a probe which must be positioned, the probe is built around the decoupling capacitors, which will naturally have most of the high-frequency (e.g.: clock edge) currents flowing through them. It is also trivial to report the measurement setup in a repeatable manner, requiring the following three characteristics: the part number of the decoupling capacitor, type of wire used, and number of wraps around the capacitor.

To acquire the data, it is necessary to use an ADC which is perfectly synchronized to the clock of the device under test. This relaxes the requirement of a high sample rate, allowing low-cost ADCs to be used for side-channel analysis. In addition, the complete design is released as an open-source project, making it available for use by researchers at [13].

There are several main areas of future work to which this capture board can be applied. First, the capture hardware can be extended to support more features. If the device under attack does not provide an accessible clock, a form of 'clock recovery' would be needed, where an adjustable 'local oscillator' is locked to the remote clock. This would require the addition of a Voltage Controlled Oscillator (VCO), which can be connected in a Phase Lock Loop (PLL) circuit. This would lock onto the pulses in the power traces which are occurring at the clock edge. Secondly, the OpenADC can be used as part of a hardware implementation of attacks. Attacks could be implemented on the FPGA itself: rather than sending the traces to the computer, they would simply be processed in real-time. This real-time processing would simplify attacks which require a considerable amount of traces, since there is no requirement to store them as an intermediate step. Finally, experimentation into different analog front-end processing, such as different filters, can easily be performed with the OpenADC.

Acknowledgments. Funded in part by NSERC Canada Graduate Scholarship. Thanks to Pankaj Rohatgi of *Cryptography Research Inc.*, and Akashi Satoh of *National Institute of Advanced Industrial Science and Technology (AIST)* for donation of the SASEBO-GII used in this work.

References

1. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
2. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
3. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)

4. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
5. Standaert, F.-X., Archambeau, C.: Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 411–425. Springer, Heidelberg (2008)
6. Jun, B., Kenworthy, G.: Is your mobile device radiating keys? In: RSA Conference 2012 (2012)
7. De Mulder, E.: Electromagnetic Techniques and Probes for Side-Channel Analysis on Cryptographic Devices. PhD thesis, KU Leuven (2010)
8. Mateos, E., Gebotys, C.: Side channel analysis using giant magneto-resistive (gmr) sensors. In: International Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE (2011)
9. Duc, G., Guilley, S., Sauvage, L., Flament, F., Nassar, M., Selmane, N., Danger, J.L., Graba, T., Mathieu, Y., Renaud, P.: Results of the 2009-2010 "dpa contest v2". In: International Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE (February 2011)
10. Carluccio, D.: Electromagnetic Side Channel Analysis of Embedded Crypto Devices. PhD thesis, Ruhr University Bochum (2005)
11. Souissi, Y., Danger, J., Guilley, S., Bhasin, S., Nassar, M.: Embedded systems security: An evaluation methodology against side channel attacks. In: 2011 Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 1–8. IEEE (2011)
12. Agilent Technologies: Triggering Wide-Bandwidth Sampling Oscilloscopes for Accurate Displays of High-Speed Digital Communications Waveforms (2005)
13. O'Flynn, C.: Openadc (2012), `http://www.newae.com/openadc`
14. Smith, D.: Signal and noise measurement techniques using magnetic field probes. In: 1999 IEEE International Symposium on Electromagnetic Compatibility, vol. 1, pp. 559–563. IEEE (1999)
15. Katashita, T., Satoh, A., Kikuchi, K., Nakagawa, H., Aoyagi, M.: Evaluation of dpa characteristics of sasebo for board level simulations. In: International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE), pp. 36–39 (2010)
16. Danis, A., Ors, B.: Differential power analysis attack considering decoupling capacitance effect. In: European Conference on Circuit Theory and Design, ECCTD 2009, pp. 359–362. IEEE (2009)
17. Faraday, M.: Experimental researches in electricity. Phil. Trans. R. Soc. Lond. 122, 125–162 (1832)
18. Weaver, J., Horowitz, M.: Measurement of via currents in printed circuit boards using inductive loops. In: 2006 IEEE Electrical Performance of Electronic Packaging, pp. 37–40. IEEE (2006)
19. Weaver, J., Horowitz, M.: Measurement of supply pin current distributions in integrated circuit packages. In: 2007 IEEE Electrical Performance of Electronic Packaging, pp. 7–10. IEEE (2007)

20. Li, L., Kim, J., Wang, H., Wu, S., Takita, Y., Takeuchi, H., Araki, K., Fan, J.: Measurement of multiple switching current components through a bulk decoupling capacitor using a lab-made low-cost current probe. In: 2011 IEEE International Symposium on Electromagnetic Compatibility (EMC), pp. 417–421. IEEE (2011)
21. Moradi, A., Mischke, O., Eisenbarth, T.: Correlation-Enhanced Power Analysis Collision Attack. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 125–139. Springer, Heidelberg (2010)
22. Satoh, A.: Side-channel attack standard evaluation board (sasebo) - dpa contest (2011), `http://www.morita-tech.co.jp/SASEBO/en/index.html`
23. Standaert, F.-X., Malkin, T.G., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)
24. Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M., Shalmani, M.T.M.: On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 203–220. Springer, Heidelberg (2008)

Appendix A: Examples of Decoupling Capacitor Measurements

Where the SASEBO-GII board provides provisions for using a current shunt power measurement, commercial boards typically provide no such considerations. In addition commercial boards often contain extensive decoupling capacitors which dampen the signal measured with a current shunt. As an example the board shown in Fig. 11 has been modified to add a current shunt along with an inductive pickup wrapped around one of the decoupling capacitors. The power trace is shown in figure Fig. 12 measured with the current shunt, and also the inductive wrapping. Note that the average of many traces shows the current

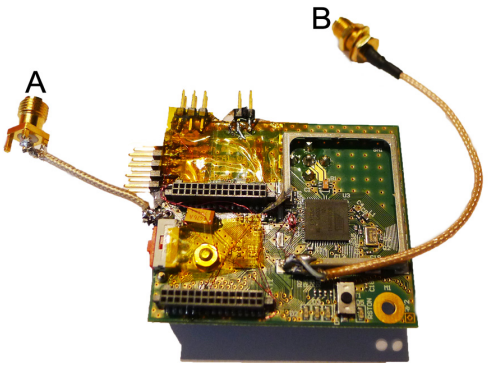


Fig. 11. Microcontroller board which supports a classic resistive shunt measurement at *(A)* and an EM measurement using the inductive wrap at *(B)*

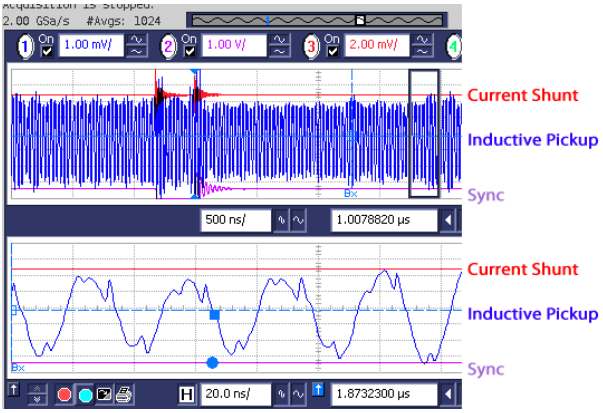


Fig. 12. A comparison of current shunt and inductive pickup on a commercial board with extensive and power supply decoupling. The program being executed switches from performing lower power load immediate instructions to higher power multiply instructions at time 1875 nS from the trigger. Red (top) is current shunt, blue (middle) is inductive pickup. Lower half of figure shows zoomed in area from black box in top half.

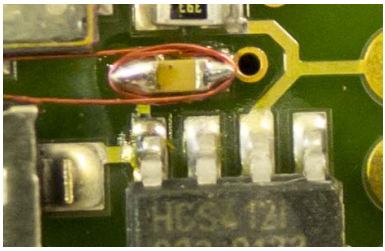


Fig. 13. Analyzing a small security device

shunt contains no visible signal, where the inductive wrap has picked up a very clean signal corresponding to different instructions being executed.

Another example of using the inductive wrapping is with small keyfob transmitters, such as those using the KeeLoq algorithm, which have been shown to be vulnerable to power analysis [24]. The decoupling capacitor is easily identified in Fig. 13, and a probe can be built around it.

Appendix B: Physical Considerations of Wrapping

A side-view of a typical SMD solder joint is shown in Fig. 14. It should be apparent that wrapping a fine magnet wire around this will be difficult, as the shape of the fillet will naturally push the wire up and off the capacitor as it is tightened. The author used a low-temperature soldering iron to put spikes towards the top side of the SMD joint. These spikes had the effect of providing a 'core' around which to wrap the magnet wire.

This process can also be assisted by using a portable dispenser for the wire - for example the Verowire Pen. The choice to solder one end of the wire to the negative pad of the decoupling capacitor results in better mechanical stability, and simplifies connection of the coaxial cable. Extensive testing showed this resulted in a very clean signal; however all the boards tested had full ground planes. One should verify the ground connection of the capacitor will provide a clean reference path.

Fig. 14. Winding the pickup coil around an arbitrary decoupling capacitor is achieved by adding a form with solder. In addition the choice to connect one side of the coil to the ground of the circuit simplifies connection of the oscilloscope in some situations.

Towards Modelling Adaptive Attacker's Behaviour*

Leanid Krautsevich[1,2], Fabio Martinelli[2], and Artsiom Yautsiukhin[2]

[1] Department of Computer Science, University of Pisa,
Largo B. Pontecorvo 3, Pisa 56127, Italy
krautsev@di.unipi.it
[2] Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche,
Via G. Moruzzi 1, Pisa 56124, Italy
{fabio.martinelli,artsiom.yautsiukhin}@iit.cnr.it

Abstract. We describe our model for the behaviour of an attacker. In the model, the attacker has uncertain knowledge about a computer system. Moreover, the attacker tries different attack paths if initially selected ones cannot be completed. The model allows finer-grained analysis of the security of computer systems. The model is based on Markov Decision Processes theory for predicting possible attacker's decisions.

Keywords: Attacker Model, Attack Graphs, Markov Decision Process.

1 Introduction

Most methods for the analysis of security of computer systems (e.g., networks, Cloud, etc.) consider attackers as omniscient entities which know all weaknesses of a computer system [4,11]. In addition, attackers are frequently assumed to make only right decisions during an attack and to exploit only the best possible way for the attack.

In contrast, descriptions of real complex attacks (e.g., [7]) show, that attackers have limited knowledge of a target system and explore the system step by step during the attack. Attackers make mistakes in their reasoning about the system, and search for alternative ways to compromise the system when the initially selected attack fails. This means that the model of powerful attacker does not provide a real description of a situation, but prepares for a worst case scenario. In reality, security teams have a limited budget and would like to concentrate on the most important security issues that can be solved within the budget. Taking into account attacker's behaviour properly is important because some attacks may be even not considered by the attacker because of her uncertain knowledge about the system or lack of resources. Wasting the budget on preventing such attacks is not the most cost-effective decision.

In this paper, we strive for a more refined attacker model introducing the attacker's view of a system, which is sometimes different from the real system.

* This work was partly supported by EU-FP7-ICT NESSoS and 295354 SESAMO projects.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 357–364, 2013.

This view drives the actions of the attacker depending on the knowledge and resources the attacker possesses. Moreover, in our model an attacker may give up on her current attack and follow an alternative attack path. We use Markov Decision Process (MDP) to model the behaviour of attacker as the method for the selection of attack steps.

The rest of the paper is organised as follows. Section 2 explains our concerns on uncertain knowledge of an attacker about a system. Section 3 focuses on models of attacker's behaviour. Related work is presented in Section 4. Section 5 concludes the paper.

2 A System and an Attacker

We consider a computer system as an attack graph G that represents the ways to compromise the system [4,11]. A node $s_i \in S$ of the attack graph denotes a successfully exploited vulnerability and an edge $a_{ij} \in A$ denotes further possible exploitation of vulnerability s_j after previously exploited vulnerability s_i. Thus, successful exploitation of vulnerabilities leads an attacker to new states with new privileges.

Similarly to our previous work [3] we group attackers into attacker profiles $\mathcal{X} = \{\Gamma, goal, intang, tang, skill\}$ where Γ is the set of attacks $\gamma \in \Gamma$ known by the attacker, *goal* is the goal of the attacker, *intang* is an amount of intangible resources possessed by the attacker, e.g., time, *tang* is the amount of tangible resources possessed by the attacker, *skill* defines how trained is the attacker. We modify the attack graph to capture properties of the attacker. First, we add to the graph an initial node corresponding to initial privileges of the attacker. Second, we define the goal nodes in the attack graph G that correspond to vulnerabilities that complete the attack (the ultimate step of each attack).

We assume that the attacker has certain amount of time units to perform the attacks. She spends a unit of time for executing a single attack step. The attacker stays in a goal state if she reaches it before spending all units of time. This situation is modelled by adding edges that start and end in the same goal state.

We separate the *real system* and the *attacker belief* about the system. When the attacker is omniscient, her view of the system coincides with the real system. We consider a more realistic case, when the view does not coincide with the real systems. The attacker's knowledge about the system determines the set of vulnerabilities that the attacker believes present in the system. These believed vulnerabilities define a new graph G_B. This graph is similar to the attack graph for the real system while has believed vulnerabilities as nodes:

$$G_B = (S_B, A_B) : S_B = S_{true} \cup S_{false}, A_B = A_{true} \cup A_{false} \tag{1}$$

where $S_{true} \subseteq S$ and $A_{true} \subseteq A$ are the subset of vulnerabilities and the subset of attack steps really existing in the system and also believed by the attacker to exist, S_{false} and A_{false} are the set of vulnerabilities and the set of action that are believed to exist but are absent in reality.

The set of vulnerabilities that are believed by the attacker is further reduced according to attacker's skills and tangible resources. Finally, the attacker has her own *view* (a graph $G_{\mathcal{X}}$) of the system:

$$G_{\mathcal{X}} = (S_{\mathcal{X}}, A_{\mathcal{X}}) : S_{\mathcal{X}} \subseteq S_B, A_{\mathcal{X}} \subseteq A_B \tag{2}$$

We assume that the system behaves probabilistically. We introduce probability $\mathbf{Pr}_{ij}$ of system transition from state i to state j in response to an attacker action. For the attacker this probability is:

$$\mathbf{Pr}_{ij} = \mathbf{Pr}^{p}_{ij} \cdot \mathbf{Pr}^{exp}_{ij} \tag{3}$$

where $\mathbf{Pr}^{p}_{ij}$ is the probability that the vulnerability j presents in the systems and $\mathbf{Pr}^{exp}_{ij}$ is the conditional probability that the vulnerability may be successfully exploited in case it exists in the system. The probability $\mathbf{Pr}_{ij}$ depends only on the successive state j while we use both indexes i and j for the uniformity with usual definition of transition probabilities.

We measure $\mathbf{Pr}^{p}_{ij}$ assuming that the attacker knows which software is installed in the system but may not know whether the software is patched or it is not. The probability of presence of the vulnerability in the system depends on the period passed after the vulnerability was discovered: the more time passed since the discovery the lower the probability of presence of the vulnerability [8]. We assume $\mathbf{Pr}^{p}_{ij}$ decreases linearly in time:

$$\begin{aligned} \mathbf{Pr}^{p}_{ij} &= -\tfrac{1}{T_{patch}} \cdot t + 1 \quad \text{if } T_{patch} \geq t \\ \mathbf{Pr}^{p}_{ij} &= 0 \qquad\qquad\quad \text{if } T_{patch} < t \end{aligned} \tag{4}$$

where T_{patch} is the time required for patching all systems, t is time passed since the release of a patch and for $t > T_{patch}$ we assume that all systems are patched. The probability $\mathbf{Pr}^{exp}_{ij}$ may be computed on the basis of score from vulnerability databases similarly to [2] or by security experts. Our approach does not depend on the method of computation of $\mathbf{Pr}^{p}_{ij}$ and $\mathbf{Pr}^{exp}_{ij}$, thus, any other methods can be used.

Example 1. We consider a company which saves information in an on-line database service. A competitor company would like to steal the information by attacking the server where the database is installed. The server operates FreeBSD 7 and MySQL 5. The database is managed by an administrator that uses a local workstation operated by Linux Mint 12 with Pidgin Messenger installed. Moreover, the administrator manages the database from her home laptop using a VPN connection to the workstation. The laptop runs Windows 7, Chrome browser, and TUKEVA Password Reminder. The whole system is depicted in Figure 1a.

The attacker composes the following attacks to the system[1]:

- The shortest possible attack requires registration in the on-line database service and execution of vulnerability CVE-2012-0484 in MySQL.

[1] Please, follow `http://nvd.nist.gov/home.cfm` for details of vulnerabilities.

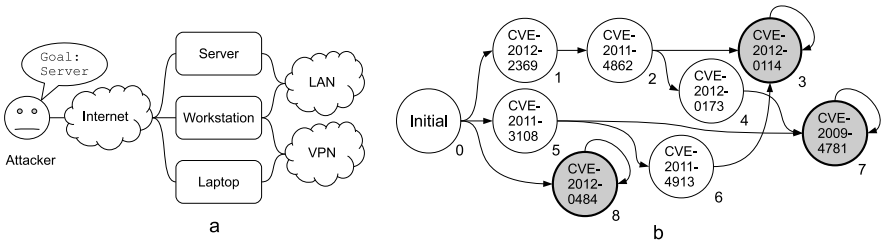


Fig. 1. a) the network system, b) the attack graph of the network system

- Another possible attack is based on vulnerability CVE-2011-3108 in Chrome browser and CVE-2009-4781 in TUKEVA where the administrator saves passwords to a database management tool.
- The attacker exploits CVE-2012-2369 in Pidgin gaining the access to the workstation. Then she causes a buffer overflow on the server using CVE-2011-4862 and exploits CVE-2012-0114 against MySQL.
- Since the laptop is connected by VPN to the workstation, the attacker gains the access to the laptop executing CVE-2012-0173 in Windows 7. Then she exploits CVE-2009-4781 in TUKEVA.
- The attacker may gain the access to the workstation after successful attack to the laptop by executing CVE-2011-4913 in the Linux kernel. Then she exploits CVE-2011-4862 on the FreeBSD server.

The resulted attack graph is displayed in Figure 1b. We enumerate the nodes for the sake of convenience. The node s_0 is the initial node. The nodes n_3, n_7 and n_8 (coloured in grey) are goal nodes.

3 Model of Attacker's Behaviour

We use Markov Decision Process (MDP) [9] to model decision making process of attackers. An attacker observes a system and can influence the behaviour of the system by making actions at moments of time (decision epochs). The system responds to an action probabilistically. The attacker decides about further actions blindly taking into account past, current, and possible future states of the system and also possible rewards that are connected with the actions. The goal of the attacker is to maximise the expected total reward (e.g., money) according to some criterion.

Formally, MDP is a set $\mathcal{P} = \{S, A, P, R, T\}$ where S is a set of system states s_i, A is a set of sets A_i of actions $a_{ij} \in A_i$ available for the attacker in the state s_i, P is a set probabilities $\mathbf{Pr}_{ij}$ that the system transits from state s_i to s_j in response to attacker's action a_{ij}, R is a set of rewards functions r_{ij} dependent on the state s_i and the action a_{ij}, T is a set of decision epochs (moments of time) t. Regarding transition probabilities, in general, the system may transit to any state available from s_i in response to the action a_{ij}. We assume that the

system only transits to the state s_j with probability $\mathbf{Pr}_{ij}$ or stays in the state i with probability $1 - \mathbf{Pr}_{ij}$.

We model attacker's behaviour as an MDP policy π which determines how an attacker selects actions. The policy is composed of decision rules. A decision rule is a procedure for the selection of an action. The attacker always selects the same action in a state if the rules are deterministic. She selects any available action at random if the rules are probabilistic.

A total reward u_π obtained by the attacker as a result of the execution of policy π is computed on the basis of instant and terminal rewards:

$$u_\pi = \sum_{t=1}^{N-1} r^t(s^t, a^t) + r^N(s^N) \tag{5}$$

where $r^t(s^t, a^t)$ is an instant reward that depends on s^t and a^t, $r^N(s^N)$ is the terminal reward that depends on the state s^N of the process at the last decision epoch N. Note, that we use upper index (e.g., s^t for a state) to denote the current value of a variable at a moment of time.

Deterministic Attacker. The simplest model of attackers behaviour [4,11,12] may be defined by an optimal deterministic policy of MDP. In this case, an attacker always prefers the best possible action in a state which belongs to the optimal attack path in the attack graph. The algorithm for the computation of optimal deterministic policies is the backward induction [9]. The algorithm finds sets $A^*_{s^t,t}$ of actions that maximise the expected total reward of the attacker.

Adaptive Attacker. We modify the behaviour of the deterministic attacker so that she may reconsider her course of action when she cannot complete her current attack path (see Algorithm 1). We assume that the attacker sets $\mathbf{Pr}^p_{ij} = 0$ (and $\mathbf{Pr}_{ij} = 0$) when she cannot complete an attack step a_{ij} and understands that the vulnerability s_j is absent in the system. In addition, the attacker sets $\mathbf{Pr}_{kj} = 0$ for all other edges entering s_j from all states s_k. Then the attacker uses the backward induction algorithm to compute a new strategy using the updated attack graph and the amount of decision epochs τ left after the initial part of the attack.

The attacker sets $\mathbf{Pr}_{kj} = \mathbf{Pr}^{exp}_{kj}$ for all edges entering s_j from all states s_k if she cannot complete the action a_{ij} but understands that the vulnerability s_j exists in the system. Then the attack strategy is recomputed according to the backward induction algorithm with the rest τ of the decision epochs. If the attacker successfully exploits the vulnerability s_j she adds edges a_{0q} and sets $\mathbf{Pr}_{0q} = \mathbf{Pr}_{jq}$ for all states s_q reachable from s_j in one step. This modification is required to remember the privileges gained by the attacker for future adjustments in her strategy.

Exploitation of Algorithm 1 allows running a simulation of interactions of an attacker and a system. We suppose that the security metrics should be estimated on the basis of several simulations (similarly to [4]).

Example 2. Let the attacker have $\tau = N$ decision epochs. She gets terminal rewards ($10K) if she reaches states 3, 7, 8, i.e., $r^N(s_3) = r^N(s_7) = r^N(s_8) = 10$ other terminal rewards equal 0. Instant rewards also equal 0. Due to space limitations we skip the computation of deterministic policies. For the attack graph presented in Figure 1, the policy is $\pi = (a^1 = a_{08})$ at the initial state during the first decision epoch. Suppose, the action is unsuccessful because the vulnerability was timely patched by the administrator. The attacker sets $\mathbf{Pr}_{08} = 0$, reconsiders initial policies using $\tau = N - 1$ decision epochs, and obtains a new policy $\pi = (a^1 = a_{05})$.

4 Related Work

There are several works which use attack graphs for the analysis of a system [11,12]. Sheyner et al. [11] determine possible attacks to the system on the basis of attack graph assuming deterministic attackers behaviour. LeMay et at. [4] proposed a work that considers attack planning where successful execution of an exploit is uncertain. In contrast to our work, the authors assume that the attacker has complete knowledge about the system and always selects the same path to her goal during the attack. Several attacker models for the analysis of

Algorithm 1. Model of adaptive attacker

```
τ := N {number of decision epochs}
t := 1 {current decision epoch}
Run the backward induction algorithm using τ to obtain A*_{s^t,t}
while τ ≠ 0 do
  if a^t = a_ij is successful then
    for all q, Pr_jq ≠ 0 do
      Pr_0q := Pr_jq
    end for
    τ := τ − 1
    t := t + 1
  else
    if s_j exists then
      for all k, Pr_kj ≠ 0 do
        Pr_kj := Pr^exp_kj
      end for
    else
      for all k, Pr_kj ≠ 0 do
        Pr_kj := 0
      end for
    end if
    τ := τ − 1
    t := 1
    Run the backward induction algorithm using τ to obtain A*_{s^t,t}
  end if
end while
```

cryptographic protocols assume that attacker may select alternative ways for compromising a system [1,5,6]. In these models, attackers know the system, i.e., the protocol, while have bounded resources. Those resources are different from ours, e.g., computational power and message manipulation capabilities.

The paper of Sarraute et al. [10] proposes to use Partially Observable Markov Decision Processes for attack planning during penetration tests. The authors analyse the system considering network configuration graph, while we consider an attack graph. In terms of knowledge collecting, authors introduce special actions that allow scanning network hosts. While we provide a way to update the graph as a result of successful and unsuccessful attack steps, and adjust the reasoning during the attack.

5 Conclusion

This work presented our initial ideas on modelling the behaviour of an attacker. We think, such approach is important if we would like to get versatile analysis of our system and protect it in the most efficient way. In particular, we considered an attacker which does not know every detail about the system, but gains the knowledge step by step. In addition, we made the attacker more flexible, i.e., the attacker is able to re-consider her plans when the initial ones fail.

As for future work we would like to incorporate the notion of decreasing attackers resources in our model as penalties of MDP. Moreover, within our approach we cannot take into account zero-day vulnerabilities, but we believe that some statistical methods could be used to tackle zero-day vulnerabilities at least approximately. Finally, we aim at creating a software prototype to evaluate system security on the basis of different metrics and versus various attacker types.

References

1. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE TIT 29, 198–208 (1983)
2. Gallon, L., Bascou, J.-J.: Cvss attack graphs. In: SITIS (2011)
3. Krautsevich, L., Martinelli, F., Yautsiukhin, A.: Formal Analysis of Security Metrics and Risk. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 304–319. Springer, Heidelberg (2011)
4. LeMay, E., Ford, M.D., Keefe, K., Sanders, W.H., Muehrcke, C.: Model-based security metrics using adversary view security evaluation (advise). In: QEST (2011)
5. Marchignoli, D., Martinelli, F.: Automatic Verification of Cryptographic Protocols through Compositional Analysis Techniques. In: Cleaveland, W.R. (ed.) TACAS/ETAPS 1999. LNCS, vol. 1579, pp. 148–162. Springer, Heidelberg (1999)
6. Mitchell, J.C., Ramanathan, A., Scedrovb, A., Teaguea, V.: A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. TCS 353, 118–164 (2006)
7. Mitnik, K.D., Simon, W.L.: The Art of Intrusion: The Real Stories Behind the Exploits of Hackers, Intruders and Deceivers. Wiley (2005)

8. Pettersen, Y.N.: Renego patched servers: A long-term interoperability time bomb brewing (July 20, 2012), http://my.opera.com/yngve/blog/2010/06/02/renego-patched-servers-a-long-term-interoperability-time-bomb-brewing
9. Puterman, M.L.: Markov Decision Processes Discrete Stochastic Dynamic Programming. Wiley-Interscience (2005)
10. Sarraute, C., Buffet, O., Hoffmann, J.: Pomdps make better hackers: Accounting for uncertainty in penetration testing. In: AAAI (2012)
11. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: IEEE SSP, pp. 273–284 (2002)
12. Wang, L., Liu, A., Jajodia, S.: Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. CC 29, 2917–2933 (2006)

Scalable Deniable Group Key Establishment

Kashi Neupane[1], Rainer Steinwandt[2,⋆], and Adriana Suárez Corona[2,⋆⋆]

[1] Atlanta Metropolitan State College, Atlanta, GA 30310
kneupane@atlm.edu
[2] Florida Atlantic University, Boca Raton, FL 33431
{rsteinwa,asuarezc}@fau.edu

Abstract. The popular Katz-Yung compiler from CRYPTO 2003 can be used to transform unauthenticated group key establishment protocols into authenticated ones. In this paper we present a modification of Katz and Yung's construction which maintains the round complexity of their compiler, but for 'typical' unauthenticated group key establishments adds authentication in such a way that deniability is achieved as well. As an application, a deniable authenticated group key establishment with three rounds of communication can be constructed.

Keywords: Group key establishment, Deniability.

1 Introduction

To simplify the design process for a group key establishment protocol, it can be convenient to restrict first to a scenario with a passive adversary, where the problem of authenticating protocol participants does not need to be addressed. Once the protocol has been proven secure in such a setting, a generic construction by Katz and Yung from CRYPTO 2003 allows to achieve security against an active adversary at the cost of one additional round [9]. Basically, this compiler appends nonces, along with sender and receiver identifiers, to all protocol messages and signs all messages with a strongly unforgeable signature scheme. This intuitive construction is round-efficient, but problematic when deniability is added as a design goal: unforgeable signatures in a protocol transcript would have to be explained in a way which does not involve the signing party.

In the two-party setting, deniability has been studied by Di Raimondo et al. [7] and Yao and Zhao [11], for instance. The problem of formalizing deniable key establishment in the group setting has been addressed in [4], where a four-round solution in the random oracle model is presented. In [12], Zhang et al. suggest an alternative formalization of deniability along with a three-round protocol in the standard model. Deniable group key establishment in a setting where the computational power of protocol participants differs is addressed by Chen et al.

⋆ RS was supported by the Spanish *Ministerio de Economía y Competitividad* through the project grant MTM-2012-15167.

⋆⋆ ASC was supported by project MTM2010 - 18370 - C04- 01 and FPU grant AP2007-03141, cofinanced by the European Social Fund.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 365–373, 2013.

[6] with a proposal in the random oracle model. Compared to [4] and [12], our definition of deniability limits the adaptivity of the adversary in corrupting users, but unlike [12] we give the adversary access to oracles that reveal session keys and send individual messages, and we do not include secret keys of corrupted users in the simulator's input. From a practical point of view this formalization of deniability seems acceptable, and we present the first general compiler to construct authenticated and deniable group key establishment protocols from 'typical' passively secure constructions.

The compiler we suggest builds on the Katz-Yung construction, but replaces the signature scheme with a suitable use of a ring signature, a message authentication code, and a multiparty key encapsulation. Like the original Katz-Yung construction, our compiler is capable of augmenting every passively secure group key establishment to an actively secure one by adding one more round. Moreover, if the unauthenticated protocol does not make use of long-term secrets—which one would typically expect—the protocol output by our compiler is deniable. In particular, applying our compiler to an unauthenticted two-round protocol as the one described in [9], which builds on work of Burmester and Desmedt [5], results in a deniable and authenticated three-round protocol.

2 Preliminaries

As main technical tools, we will make use of a multiparty key encapsulation along with a suitable data encapsulation mechanism to send an identical message to multiple protocol participants in a confidential manner. For implementing the authentication we also make use of a message authentication code and a ring signature. Here and in the subsequent sections, the security parameter will be denoted by k, and notions like polynomial time or negligible refer to k.

2.1 Multi Key Encapsulation and Symmetric Encryption

In [10], Smart introduced the notion of a *multi key encapsulation mechanism (mKEM)*, generalizing key encapsulation to a setting with multiple recipients. A group key establishment by Gorantla et al. [8] makes use of this primitive, and our treatment of mKEMs follows the latter.

Definition 1 (multi key encapsulation mechanism)
A multi key encapsulation mechanism (mKEM) *is a triple of polynomial time algorithms* (`mKeyGen`, `mEncaps`, `mDecaps`) *as follows:*

- `mKeyGen` *is probabilistic. Given the domain parameters* $\mathbb{D}$*, it generates a pair of public and secret keys* (pk, dk)*.*
- `mEncaps` *is probabilistic. Given a (polynomial size) set* $\{pk_1, \ldots, pk_n\}$ *of public keys it generates a pair* (K, C) *where* $K \in \{0,1\}^k$ *is a session key and* C *is an encapsulation of this session key under the public keys* $\{pk_1, \ldots, pk_n\}$*.*
- `mDecaps` *is deterministic. Given a secret key* dk *and an encapsulation* C*, this algorithm outputs the session key* K *or a special error symbol* $\perp$*.*

We require that for all key pairs (pk_i, dk_i) *generated by* `mKeyGen` *the implication* $(K, C) = \texttt{mEncaps}(\{pk_1, \ldots, pk_n\}) \Longrightarrow \texttt{mDecaps}_{dk_i}(C) = K$ *holds* $(i = 1, \ldots, n)$.

Informally, we consider an mKEM as IND-CCA secure if no probabilistic polynomial time adversary, with access to a decapsulation oracle, can distinguish with more than negligible probability which of two keys is encapsulated in a challenge C^* for a set of public keys of his choice—see [8] for the formal definition. To be able to actually encrypt messages in our compiler, we combine an mKEM with a suitable *data encapsulation mechanism*, which we realize as a symmetric encryption scheme offering security in the real-or-random sense (cf. Bellare et al. [1]):

Definition 2 (symmetric encryption scheme). *A symmetric encryption scheme is a triple of polynomial time algorithms* $(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ *as follows:*

- `KeyGen` *is probabilistic. Given the security parameter* 1^k*, it generates a secret key* K*.*
- `Enc` *is probabilistic. Given a secret key* K *and a message* $m \in \{0,1\}^*$*, this algorithm generates a ciphertext* C*.*
- `Dec` *is deterministic. Given a ciphertext* C *and a secret key* K*, this algorithm outputs either a message* m *or a dedicated error symbol* $\perp$*.*

We require that $m = \texttt{Dec}_K(\texttt{Enc}_K(m))$ *for all keys* K *and for all* $m \in \{0,1\}^*$.

Informally, a symmetric encryption scheme is secure in the real-or-random sense if no efficient adversary can distinguish with more than negligible probability whether the bitstring encrypted is a message m or a uniformly at random chosen bitstring of the same length as m—see Bellare et al. [1] for the formal definition.

2.2 Message Authentication Codes and Ring Signatures

To solve the problem of authentication without jeopardizing deniability, our compiler uses a message authentication code as well as a suitable ring signature.

Definition 3 (message authentication code). *A* message authentication code (MAC) *is a tuple* $(\texttt{MKeyGen}, \texttt{Tag}, \texttt{Verify})$ *of polynomial time algorithms:*

- `MKeyGen` *is probabilistic. Given the domain parameters* $\mathbb{D}$*, it generates a secret key* K*.*
- `Tag` *is probabilistic. Given a message* $m \in \{0,1\}^*$ *and a secret key* K *it generates a message tag* $\theta := \texttt{Tag}_K(m) \in \{0,1\}^*$ *on* m*.*
- `Verify` *is deterministic. Given a message* m*, a secret key* K *and a candidate tag* θ*,* `Verify` *returns* 1 *if* θ *is a valid tag for the message* m *and* 0 *otherwise.*

The compiler below assumes that the MAC we employ is strongly unforgeable under adaptive chosen message attacks (cf. [2]), i. e., no probabilistic polynomial time adversary with access to tagging and verifying oracles for key K can produce a valid (message, tag)-pair with more than negligible probability.

Finally, our compiler makes use of a ring signature scheme which enables a signer to produce signatures which can be verified successfully under several verification keys.

Definition 4 (ring signature scheme). *A* ring signature scheme *is a tuple of polynomial time algorithms* (RKeyGen, RSign, RVerify) *as follows:*

- RKeyGen *is probabilistic. Given the security parameter k, it generates a pair of keys (vk, sk), where vk is a public verification key and sk is its corresponding secret signing key.*
- RSign. *Given a message m, a polynomial size set (a* ring*) of public verification keys $\mathcal{R} = \{vk_1, \ldots, vk_n\}$ and a secret key sk_s such that $vk_s \in \mathcal{R}$, this algorithm produces a signature σ.*
- RVerify *is deterministic. Given a message m, a signature σ and a ring of public keys $\mathcal{R}$, this algorithm returns 1 if σ is a valid signature for the message m with respect to the ring $\mathcal{R}$, and 0 otherwise.*

We require that for any ring $\mathcal{R}$ comprised of public verification keys produced by RKeyGen *and for any message m the relation* $\texttt{RVerify}(m, \texttt{RSign}_{sk}(m, \mathcal{R}), \mathcal{R}) = 1$ *holds, where sk is the secret key for a verification key $vk \in \mathcal{R}$.*

For a ring signature, it is usually expected that the adversary cannot know which user in the ring was the actual signer of a message. A strong form of this design goal is known as *anonymity against full key exposure*—we refer to [3] for the formal definition. Of course, for a ring signature scheme we also expect an appropriate form of existential unforgeability. More specifically, we impose unforgeability as defined in [3], i. e., that no probabilistic polynomial time adversary having access to ring signature and private key oracles can produce a valid (message, ring signature, ring)-tuple with more than negligible probability.

3 Security Model

To formalize secure group key establishment, we follow Katz and Yung [9]. In addition to authentication and semantic security, the compiler discussed in the next section aims at the resulting protocol to be deniable. When privacy is a concern, it is desirable that the participation of a user in a protocol cannot be proved by showing a protocol transcript. Bohli and Steinwandt [4] formalized this idea in the following definition of *deniability* for group key establishment:

Deniability According to [4]. Let $\mathcal{A}_d$ denote a probabilistic polynomial time algorithm with the security parameter 1^k as input. In addition, $\mathcal{A}_d$ obtains the public information pk made available in the initialization phase as input (after application of the compiler below this includes in particular the public verification keys of the underlying ring signature scheme). Finally $\mathcal{A}_d$ obtains as input an upper bound q_c on the number of protocol participants that can be corrupted.

After having obtained this input, $\mathcal{A}_d$ interacts with protocol instances via the Corrupt, Reveal, and Send oracle as usual.[1] However, $\mathcal{A}_d$ must not query Test, and at most q_c queries to Corrupt can be submitted. Eventually, $\mathcal{A}_d$ outputs a bitstring $T_{\mathcal{A}_d} = T_{\mathcal{A}_d}(k, q_c, pk)$—which represents a protocol transcript that

[1] To simulate the Execute oracle, Send-queries can be used, so Execute can be omitted.

serves as evidence for the involvement of a particular user in the group key establishment. We denote by $T_{\mathcal{A}_d} = T_{\mathcal{A}}(k, q_c)$ the random variable that describes $T_{\mathcal{A}_d}(k, q_c, pk)$ with the randomness for $\mathcal{A}_d$, for protocol instances, and in the initialization phase, being chosen uniformly at random.

To capture deniability a second algorithm $\mathcal{S}_d$, to which we refer as *simulator*, is used: this simulator accepts the same input as $\mathcal{A}_d$ and can impose the same maximum number q_c of corrupted users as the latter. However, $\mathcal{S}_d$ is not allowed to invoke any uncorrupted user. More specifically, $\mathcal{S}_d$ can submit up to q_c queries to Corrupt, but can neither query Reveal nor Send (nor Execute nor Test). The output of $\mathcal{S}_d$ is a bitstring $T_{\mathcal{S}_d}(k, q_c, pk)$, and analogously as for $\mathcal{A}_d$ we define a random variable $T_{\mathcal{S}_d}(k, q_c)$, based on uniformly at random chosen randomness.

Definition 5 (deniability). *A group key establishment protocol is* deniable *if for every probabilistic polynomial time adversary $\mathcal{A}_d$ as specified above and every $q_c \in \mathbb{N}_0$ there is a probabilistic polynomial time simulator $\mathcal{S}_d$ such that $T_{\mathcal{A}_d}(k, q_c)$ and $T_{\mathcal{S}_d}(k, q_c)$ are computationally indistinguishable. In other words, no probabilistic polynomial time algorithm can distinguish $T_{\mathcal{A}_d}(k, q_c)$ and $T_{\mathcal{S}_d}(k, q_c)$ with non-negligible probability.*

A More Relaxed Notion of Deniability. The definition of deniability just discussed allows the adversary $\mathcal{A}_d$ to fix the corrupted parties in a fully adaptive manner. In the definition used subsequently we restrict this freedom and require $\mathcal{A}_d$ to complete all corruptions before querying Send. On the intuitive side, this materializes the idea that the parties who are willing to conspire (and reveal their secret keys to this aim) already enter protocol executions with this intent. As we still allow an arbitrary subset of the users to be corrupted, the resulting notion of deniability seems still natural and useful.

Remark 1. Unlike [12], we do not integrate authentication into the definition of deniability. Further, differing from [12], we give the adversary used in the definition of deniability full access to Send and Reveal and do not include secret keys of corrupted users in the simulator's input.

As before, let $\mathcal{A}_d$ denote a probabilistic polynomial time algorithm with the security parameter 1^k and public information pk from the initialization phase as input. No upper bound on the number of corruptions is imposed. In a first phase $\mathcal{A}_d$ has access to the Corrupt-oracle only, and can (adaptively) corrupt an arbitrary subset of the users (including the case of no user or all users being corrupted). Hereafter, in a second phase, $\mathcal{A}_d$ interacts with the protocol participants via the Reveal- and Send-oracle. Neither Corrupt nor Test may be queried in this phase. Analogously as in the definition of [4], $\mathcal{A}_d$ outputs a bitstring $T_{\mathcal{A}_d} = T_{\mathcal{A}_d}(k, pk)$ to evidence the involvement of a particular user in the group key establishement. Let $T_{\mathcal{A}_d} = T_{\mathcal{A}_d}(k)$ be the random variable describing $T_{\mathcal{A}_d}(k, pk)$ with the randomness for $\mathcal{A}_d$, for all protocol instances, and in the initialization phase being chosen uniformly at random.

The simulator $\mathcal{S}_d$ obtains the same input as $\mathcal{A}_d$, but can only access the Corrupt oracle—no access to Reveal, Send, or Test is available. The output of $\mathcal{S}_d$

is a bitstring $T_{\mathcal{S}_d}(k, pk)$, and analogously as for $\mathcal{A}_d$ we define a random variable $T_{\mathcal{S}_d}(k)$ based on uniformly at random chosen randomness. Consider the following experiment for a probabilistic polynomial time distinguisher $\mathcal{X}$ outputting 0 or 1: the challenger flips a random coin $b \in \{0,1\}$ uniformly at random. If $b = 1$, the transcript $T_{\mathcal{A}_d}(k)$ is handed to $\mathcal{X}$, whereas for $b = 0$ the transcript $T_{\mathcal{S}_d}(k)$ is handed to $\mathcal{X}$. The distinguisher $\mathcal{X}$ wins whenever the guess b' it outputs for b is correct; the advantage of $\mathcal{X}$ is denoted by $\mathrm{Adv}^{\mathrm{den}}_{\mathcal{X}} := \left|\Pr[b = b'] - \frac{1}{2}\right|$. In this paper we will use the following definition of deniability:

Definition 6 ((relaxed) deniability). *A group key establishment protocol is* deniable *if for every polynomial time adversary $\mathcal{A}_d$ as specified above there exists a probabilistic polynomial time simulator $\mathcal{S}_d$ such that the following holds:*

- *With overwhelming probability, the number of* Corrupt*-queries of $\mathcal{S}_d$ is less than or equal to the number of* Corrupt*-queries of $\mathcal{A}_d$.*
- *For each probabilistic polynomial time distinguisher $\mathcal{X}$, the advantage* $\mathrm{Adv}^{\mathsf{den}}_{\mathcal{X}}$ *in the above experiment is negligible.*

4 From Unauthenticated to Authenticated and Deniable

Subsequently we denote by (`mKeyGen`, `mEncaps`, `mDecaps`) an IND-CCA secure multi key encapsulation (see [8]) and by (`KeyGen`, `Enc`, `Dec`) a ROR-CCA secure symmetric encryption scheme (see [1]). To simplify the description, we assume that $\texttt{KeyGen}(1^k)$ simply returns a uniformly at random chosen bitstring in $\{0,1\}^k$ (alternatively one could use keys obtained from `mDecaps` to fix the randomness of `KeyGen`). By (`MKeyGen`, `Tag`, `Verify`) we denote an SUF-CMA secure message authentication code (see [2]) and by (`RKeyGen`, `RSign`, `RVerify`) an RSIG-UF secure ring signature scheme which is anonymous (see [3]).

4.1 Description of the Proposed Compiler

The proposed compiler modifies a given (semantically secure) *unauthenticated* group key establishment protocol P to obtain a protocol P' which is *authenticated*. Moreover, if the original protocol P does not make use of long-term secrets, then the resulting protocol P' is *deniable*. Further, if the original protocol is forward secure the compiled protocol preserves this property. For the ease of notation, assume that $U_0, \ldots, U_{n-1}$ are the users who want to establish a secret key. One of the protocol participants has a special role in the compiled protocol—it is the only user that will sign a message in the newly added Round 0. We refer to this user as initiator and without loss of generality assume that U_0 plays this role. Moreover, for the ease of notation, in our description, we do not explicitly refer to individual instances.

Finally, for the messages in protocol P, let $m_{i,l}^{(j)}$ be the message sent by user U_i in the j-th round to user U_l. We can without loss of generality assume that instead of sending these messages, in Round j the user U_i broadcasts the combined message $m_{i,j} = U_i||j||\left(m_{i,1}^{(j)}, \ldots, m_{i,n-1}^{(j)}\right)$. In particular, $m_{i,j}$ includes an

(unprotected) identifier of the sender U_i of the message and of the round number. Each recipient U_l can recover $m_{i,l}^{(j)}$ in the obvious way, and this change does neither affect the security nor the round complexity of P.

Initialization Phase. In addition to the initialization for protocol P, each user U_i generates a (public key, secret key)-pair (pk_i, dk_i) for the above-mentioned multi key encapsulation scheme, and the public keys are made available to all users (and the adversary). Similarly, each user U_i generates a (verification key, signing key)-pair (vk_i, sk_i) for the before-mentioned ring signature scheme.

Next, our compiler adds a new round to protocol P as follows:

Introduction of Round 0. In this new initial round, each user U_i $(i \neq 0)$ chooses a random nonce $r_i \in \{0,1\}^k$ and broadcasts $U_i||0||r_i$. The initiator U_0 performs the following steps:

- run $\mathtt{MKeyGen}$ to generate a key K_0 for the message authentication code;
- produce a ring signature $\sigma := \mathtt{RSig}_{sk_0}(K_0||\mathsf{pid}_0||U_0||0||r_0, \mathsf{pid}_0)$;
- compute $(K, C) \leftarrow \mathtt{mEncaps}(\mathsf{pid}_0)$;
- produce a ciphertext $E := \mathtt{Enc}_K(K_0||\mathsf{pid}_0||U_0||0||r_0||\sigma)$ using the symmetric key K;
- compute a tag $\mathsf{tag}_0 = \mathtt{Tag}_{K_0}(C, E)$, and
- broadcast $U_0||0||(C, E)||\mathsf{tag}_0$.

After receiving the Round 0 message of all parties, each user U_i executes the following steps:

- set $\mathsf{nonces}_{U_i} = ((U_1, r_1), \cdots, (U_n, r_n))$ and store this value;
- run $\mathtt{mDecaps}_{dk_i}(C)$ to obtain K;
- decrypt the ciphertext E using K;
- verify the ring signature for the ring pid_i and the tag tag_0; if the verification fails or if $\mathsf{pid}_0 \neq \mathsf{pid}_i$, the protocol is aborted.

Now, in each original round of P we use K_0 for authentication as follows:

Modification of Round j, $j \neq 0$. If the protocol is not aborted, if user U_i is supposed to broadcast $m_{i,j}$ in protocol P, then U_i will instead do the following:

- use K_0 to compute a tag $\mathsf{tag}_{i,j} = \mathtt{Tag}_{K_0}(m_{i,j}||\mathsf{nonces}_{U_i})$.
- broadcasts $m_{i,j}||\mathsf{tag}_{i,j}$.

When receiving a message $m_{l,j}||\mathsf{tag}_{l,j}$, user U_i checks the following:

- $U_l \in \mathsf{pid}_0$
- j is the expected round number
- Verify the tag $\mathsf{tag}_{l,j}$.

If any of these checks fails, the protocol is aborted without accepting a session key. Otherwise, the session identifier is the concatenation of all messages sent and received by the protocol instance during its execution and the session key is as in P.

Remark 2. With a slight abuse of notation, here we identify a partner identifier pid_j with the set of public keys of the users contained in this partner identifier.

4.2 Security Analysis

Making no further assumptions about the protocol P, we have the following result, which says that the above compiler adds authentication as desired. Because of the page limit, a proof of this result is not included here.

Proposition 1. *With the above notation, the group key establishment obtained from the compiler in Section 4.1 is authenticated and secure in the sense of [9] (in particular, forward security is preserved).*

In principle we can apply the compiler in Section 4.1 to some fully authenticated protocol, which signs all messages sent by parties. In such a case we cannot expect that the compiled protocol is deniable. The more typical passively secure protocol does not involve any long-term secrets, and in such a setting the proposed compiler does ensure deniability. Again, because of the page limit we do not include a proof here.

Proposition 2. *If the group key establishment* P *does not involve long-term secrets, then the group key establishment* P' *obtained by applying the compiler in Section 4.1 to* P *is deniable in the sense of Definition 6.*

5 Conclusion

Given an unauthenticated group key establishment, the above protocol compiler outputs an authenticated group key establishment with one additional round. Provided that the given unauthenticated protocol does not involve long-term secrets, the resulting protocol is also deniable. So the compiler seems an interesting alternative to the popular Katz-Yung construction, if privacy guarantees are a concern.

References

1. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. Full paper of an extended abstract that appeared in the Proceedings of the 38th Symposium on Foundations of Computer Science. IEEE (August 1997)
2. Bellare, M., Namprempre, C.: Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
3. Bender, A., Katz, J., Morselli, R.: Ring Signatures: Stronger Definitions, and Constructions Without Random Oracles. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 60–79. Springer, Heidelberg (2006)
4. Bohli, J.-M., Steinwandt, R.: Deniable Group Key Agreement. In: Nguyen, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 298–311. Springer, Heidelberg (2006)
5. Burmester, M., Desmedt, Y.: A Secure and Efficient Conference Key Distribution System. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 275–286. Springer, Heidelberg (1995)

6. Chen, S., Cheng, Q., Ma, C.: A Deniable Group Key Exchange Protocol for Imbalanced Wireless Networks. In: Hu, B., Li, X., Yan, J. (eds.) 5th International Conference on Pervasive Computing and Applications (ICPCA 2010), pp. 1–5. IEEE (2010)
7. Di Raimondo, M., Gennaro, R., Krawczyk, H.: Deniable Authentication and Key Exchange. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, pp. 400–409. ACM (2006)
8. Choudary Gorantla, M., Boyd, C., González Nieto, J.M., Manulis, M.: Generic One Round Group Key Exchange in the Standard Model. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 1–15. Springer, Heidelberg (2010)
9. Katz, J., Yung, M.: Scalable Protocols for Authenticated Group Key Exchange. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 110–125. Springer, Heidelberg (2003)
10. Smart, N.P.: Efficient Key Encapsulation to Multiple Parties. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 208–219. Springer, Heidelberg (2005)
11. Yao, A.C., Zhao, Y.: Deniable Internet Key Exchange. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 329–348. Springer, Heidelberg (2010)
12. Zhang, Y., Wang, K., Li, B.: A Deniable Group Key Establishment Protocol in the Standard Model. In: Kwak, J., Deng, R.H., Won, Y., Wang, G. (eds.) ISPEC 2010. LNCS, vol. 6047, pp. 308–323. Springer, Heidelberg (2010)

Information-Theoretic Foundations of Differential Privacy*

Darakhshan J. Mir

Rutgers University, Piscataway NJ 08854, USA
mir@cs.rutgers.edu

Abstract. We examine the information-theoretic foundations of the increasingly popular notion of *differential privacy*. We establish a connection between differential private mechanisms and the *rate-distortion* framework. Additionally, we also show how differentially private distributions arise out of the application of the *Maximum Entropy Principle*. This helps us locate differential privacy within the wider framework of information-theory and helps formalize some intuitive aspects of our understanding of differential privacy.

1 Introduction

The problem of releasing aggregate information about a statistical database while simultaneously providing privacy to the individual participants of the database has been extensively studied in the computer science and statistical communities. *Differential privacy* (DP) has been one of the main lines of research that has emerged out of attempts to formalize and solve this problem, over the last few years. See [5] for a survey. It formalizes the idea that privacy is provided if the "identification risk" an individual faces does not change appreciably if he or she participates in a statistical database.

Often, in the context of data privacy, and more specifically, differential privacy, the claim is made that *privacy* and *utility* are conflicting goals. The application of differential privacy to several problems of private data analysis has made it clear that the utility of the data for a specific measurement degrades with the level of privacy. The greater the level of privacy, the less "useful" the data is, and vice versa. This paper attempts to understand the precise information-theoretic conditions that necessitate such a trade-off. We observe that differentially-private mechanisms arise out of minimizing the information leakage (measured using information-theoretic notions such as mutual information) while trying to maximize "utility". The notion of utility is captured by the use of an abstract distortion function **dist** that measures the distortion between the input and the output of the mechanism. This is a general mechanism, and can be instantiated appropriately depending on the problem domain. The main observation of this paper is that the probability distribution that achieves this constrained minimization corresponds to the so-called *exponential mechanism* [11]. We also show

* This work was supported by NSF award number CCF-1018445.

J. Garcia-Alfaro et al. (Eds.): FPS 2012, LNCS 7743, pp. 374–381, 2013.

how differentially-private mechanisms arise out of the application of the *principle of maximum entropy*, first formulated by Jaynes [7]. We see that among all probability distributions that constrain the expected distortion to stay within a given value, the differentially private mechanism, corresponds to the distribution that maximizes the conditional entropy of output given the input. This, to our knowledge, is the first attempt at providing an information theoretic foundation for differential privacy. In Section 2 we review the appropriate definitions and notions from differential privacy. In Section 2.1 we discuss related work. In Sections 3 and 4 we present our main results.

2 Definitions and Background

In this section we present the background and the related work in differential privacy. Assume a probability distribution $p_{\mathbf{X}}(\mathbf{x})$ on an alphabet $\mathcal{X}$. $\mathcal{X}$ may either be a scalar or vector space. Let $\mathbf{X}_i \in \mathcal{X}$ be a random variable representing the i-th row of a database. Then the random variable representing a database of size n, (whose elements are drawn from $\mathcal{X}$) is $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2 \ldots, \mathbf{X}_n)$. $\mathbf{x}$ represents the value that the random variable $\mathbf{X}$ takes, that is the observed database $\mathbf{x}$. Note that the $\mathbf{X}$'s themselves may multi-dimensional representing the k attributes of the database. Dwork et al. [6] define the notion of differential privacy that provides a guarantee that the probability distribution on the outputs of a mechanism is "almost the same," irrespective of whether or not an individual is present in the data set. Such a guarantee incentivizes participation of individuals in a database by assuring them of incurring very little risk by such a participation. To capture the notion of a user opting in or out, the "sameness" condition is defined to hold with respect to a neighbor relation; intuitively, two inputs are neighbors if they differ only in the participation of a single individual. For example, Dwork et al. [6] define datasets to be neighbors if they differ in a single row. McGregor et. al [10] define differential privacy, equivalently, in terms of probability distributions. This formulation is more useful for us.

Definition 1. *[10] Let* $\mathbf{x}$ *be a database of length* n*, drawing each of its elements from an alphabet* $\mathcal{X}$*, then an* ε-differentially private mechanism *on* $\mathcal{X}^n$ *is a family of probability distributions* $\{\pi(\mathbf{o}|\mathbf{x}) : \mathbf{x} \in \mathcal{X}^n\}$ *on a range* $\mathcal{O}$*, such that for every neighboring* $\mathbf{x}$ *and* $\mathbf{x}'$*, and for every measurable subset* $\mathbf{o} \subset \mathcal{O}$*,* $\pi(\mathbf{o}|\mathbf{x}) \leq \pi(\mathbf{o}|\mathbf{x}') \exp(\varepsilon)$.

Notice that the distribution (or equivalently) mechanism is parametrized by the input database $\mathbf{x}$ or $\mathbf{x}'$, whichever is relevant.

One mechanism that Dwork et al. [6] use to provide differential privacy is the *Laplacian noise method* which depends on the *global sensitivity* of a function:

Definition 2. *[6] For* $f : \mathcal{X}^n \to \mathbb{R}^d$*, the global sensitivity of* f *is* $\Delta f = \max_{\mathbf{x} \sim \mathbf{x}'} ||f(\mathbf{x}) - f(\mathbf{x}')||_1$.

Another, more general (though, not always computationally efficient) method of providing differential privacy is the so called *exponential mechanism* proposed by

McSherry and Talwar [11]. This mechanism can be said to be parametrized by a "distortion function" $\mathbf{dist}(\mathbf{x}, \mathbf{o})$ that maps a pair of an input data set $\mathbf{x}$ (a vector over some arbitrary real-valued domain) and candidate output $\mathbf{o}$ (again over an arbitrary range $\mathcal{O}$) to a real valued "distortion score." Lower valued distortions imply good input-output correspondences. It assumes a base measure π on the range $\mathcal{O}$. For a given input $\mathbf{x}$, the mechanism selects an output $\mathbf{o}$ with exponential bias in favor of low distorting outputs by sampling from the following *exponential distribution* [11]:

$$\pi^{\varepsilon}(\mathbf{o}) \propto \exp(-\varepsilon\, \mathbf{dist}(\mathbf{x}, \mathbf{o})) \cdot \pi(\mathbf{o}). \tag{1}$$

Theorem 1. *[11] The exponential mechanism, when used to select an output* $\mathbf{o} \in \mathcal{O}$*, gives* $2\varepsilon\Delta\,\mathbf{dist}$*-differential privacy, where* $\Delta\,\mathbf{dist}$ *is the global sensitivity of the distortion function* **dist**.

The exponential mechanism is a useful abstraction when trying to understand differential privacy because it generalizes all specific mechanisms, such as the Laplacian mechanism introduced above. The exponential mechanism because of the generality of the input space $\mathcal{X}$, the output range $\mathcal{O}$ and the distortion function **dist**, captures all differentially private mechanisms. The π^{ε} denotes the dependence of the posterior on $\pi(\mathbf{o}|\mathbf{x})$, on the parameter ε.

2.1 Related Work

Some information-theoretic notions and metrics of data privacy exist in the literature. See [17], [3], for example. Sankar et. al [14] consider the problem of quantifying the privacy risk and utility of a data transformation in an information-theoretic framework. Rebello-Monedero [13] consider the problem in a similar framework and define an information-theoretic privacy measure similar to an earlier defined measure of *t-closeness* [8]. A connection between information theory and differential privacy through Quantitative flow has been made by Alvim et al. [1]. Alvim et al. [1] use the information-theoretic notion of Min-entropy for the information leakage of the private channel, and show that differential privacy implies a bound on the min-entropy of such a channel. They also show how differential privacy imposes a bound on the utility of a randomized mechanism and under certain conditions propose an optimal randomization mechanism that achieves a certain level of differential privacy. Barthe and Kopf [2] also develop upper bounds for the leakage of every ε-differentially private mechanism. Our work is different from (but related to) theirs in the sense that we do not aim at finding bounds for the information leakage (or risk) of the differentially-private mechanisms. Our aim is to understand the information-theoretic foundations of the framework of differential privacy. Our work is in the spirit of Sankar et al. [14] and Rebello-Monedero et al. [13] but examining how a risk-distortion tradeoff gives rise to differentially-private mechanisms. In previous work [12] we examine the information theoretic connections of differentially-private learning. This was done in a specific context of learning, and the general implications were not clear.

3 Differentially-Private Mechanisms in a Risk-Distortion Framework

Assume an input space $\mathcal{X}^n$, and a range $\mathcal{O}$. For any $\mathbf{x} \in \mathcal{X}^n$, and any output $\mathbf{o} \in \mathcal{O}$, a distortion function **dist** is specified. Consider a probability measure $p_{\mathbf{X}}(\mathbf{x})$ on $\mathcal{X}$ and a prior probability π on $\mathcal{O}$.

Given a database $\mathbf{x}$, which is a set of n random independent samples $\mathbf{X} = \{\mathbf{X}_1, \ldots \mathbf{X}_n\} \in \mathcal{X}^n$, where each $\mathcal{X}_i$ is drawn i.i.d from $p_{\mathbf{X}}(\mathbf{x})$, and an output $\mathbf{o}$, the "utility" of $\mathbf{o}$ for $\mathbf{x}$, is given by (the negative of) a function $\mathbf{dist} : \mathcal{X}^n \times \mathcal{O} \to \mathbb{R}$.

The expected distortion of a mechanism $\pi_{\mathbf{O}|\mathbf{X}}(\mathbf{o}|\mathbf{x})$ is:

$$\mathbb{E}_{\mathbf{x} \sim p_{\mathbf{X}}(\mathbf{x})^n} \mathbb{E}_{\mathbf{o} \sim \pi(\mathbf{o}|\mathbf{x})} \mathbf{dist}(\mathbf{x}, \mathbf{o}).$$

Rebollo-Monedero et. al [13] define a privacy risk function to be the mutual information between the revealed and the hidden random variables. Similarly, we define a privacy risk function $\mathcal{R}$ to be the mutual information between the input (the underlying database) and the output of the differentially private mechanism, that is, $\mathcal{R} = I(\mathbf{X}; \mathbf{O})$. We know that the mutual information

$$I(\mathbf{X}; \mathbf{O}) = H(\mathbf{O}) - H(\mathbf{O}|\mathbf{X}) = H(\mathbf{X}) - H(\mathbf{X}|\mathbf{O}), \tag{2}$$

where $H(\mathbf{X})$ represents the entropy of the random variable of $\mathbf{X}$ and $H(\mathbf{O}|\mathbf{X})$ the conditional entropy of $\mathbf{O}$ given $\mathbf{X}$. So, the mutual information is the reduction in the uncertainty about $\mathbf{X}$ by knowledge of output $\mathbf{O}$ or vice versa (See [4] for example). Also we have that

$$\mathcal{R} = I(\mathbf{X}; \mathbf{O}) = \mathbb{E} \log \frac{\pi(\mathbf{O}|\mathbf{X}) p(\mathbf{X})}{\pi(\mathbf{O}) p(\mathbf{X})} = \mathbb{E} \log \frac{\pi(\mathbf{O}|\mathbf{X})}{\pi(\mathbf{O})}. \tag{3}$$

This is equal to the conditional Kullback-Leibler divergence between the posterior and prior distributions denoted by $D_{KL}\left(\pi(\mathbf{O}|\mathbf{X}) \| \pi(\mathbf{O})\right)$. If the prior and posterior distributions are the same, then the privacy risk is zero, but that also means that the distortion may be arbitrarily high. However, we are interested in minimizing the distortion function associated with the posterior distribution, while minimizing the privacy risk $\mathcal{R}$. As a result, we are interested in quantifying this risk-distortion trade-off. Notice that untill this point, our risk-distortion framework is formulated only in information-theoretic terms. We will see how the differentially-private mechanism arises out of this framework.

As in Rebollo-Mondero et. al [13], we are interested in a randomized output, minimizing the privacy risk given a distortion constraint (or viceversa). Unlike their treatment, however, the potential outputs are more general than perturbations of the input database elements to capture differentially-private mechanims (both interactive and noninteractive). The privacy risk-distortion function is defined analogously (as in Rebollo-Mondero [13]), as

$$\mathcal{R}(\mathcal{D}) = \inf_{\pi_{\mathbf{O}|\mathbf{X}} : \mathbb{E}_{\mathbf{x},\mathbf{o}} \mathbf{dist}(\mathbf{x}, \mathbf{o}) \leq \mathcal{D}} I(\mathbf{X}; \mathbf{O}) \tag{4}$$

3.1 Connection to the Rate-Distortion Framework

Rebollo-Mondero et. al relate the risk-distortion function formulated in Equation 4 [13] to the well-known *rate-distortion* problem in information theory first formulated by Shannon. (See [4], for example). Shannon's rate-distortion theory is applied in the context of lossy compression of data. The objective is to construct a compact representation (a code) of the underlying signal (or data), such that the average distortion of the signal reconstructed from this compact representation is low. Rate-distortion theory determines the level of expected distortion $\mathcal{D}$, given the desired information rate $\mathcal{R}$ of the code or vice-versa using the rate-distortion function $\mathcal{R}(\mathcal{D})$ similar to that in Equation 4 where $\mathcal{R}$ is the information rate of the code, when applied to the compression problem. So, the rate-distortion function is defined as the infimum of the rates of codes whose distortion is bounded by $\mathcal{D}$.

Using this connection, one can prove the following:

Theorem 2. *[13] The privacy risk-distortion function is a convex and non-increasing function of $\mathcal{D}$.*

The problem is to minimize the privacy risk, defined thus, under the expected distortion constraint. As a function of the probability density, $\pi_{\mathbf{O}|\mathbf{X}}(\mathbf{o}|\mathbf{x})$, the problem is also convex. We can also use Lagrangian multipliers to write Equation 4 in an equivalent unconstrained form. We have the functional

$$\mathcal{F}[\pi(\mathbf{o}|\mathbf{x})] = \frac{1}{\varepsilon} I(\mathbf{X};\mathbf{O}) + \mathbb{E}\,\mathbf{dist}(\mathbf{X},\mathbf{O}). \tag{5}$$

for a positive ε. Functional $\mathcal{F}$ needs to be minmized among all normalized $\pi(\mathbf{o}|\mathbf{x})$. So we can find the distribution that minimizes this funtion, by using standard optimization techniques. Standard arithmetic manipulation, leads Tishby et al. [16] to prove the following theorem:

Theorem 3. *[16] The solution of the variational problem, $\frac{\partial \mathcal{F}}{\partial \pi(\mathbf{o}|\mathbf{x})} = 0$, for normalized distributions $\pi(\mathbf{o}|\mathbf{x})$, is given by the exponential form*

$$\pi^{\varepsilon}(\mathbf{o}|\mathbf{x}) = \frac{\exp(-\varepsilon\,\mathbf{dist}(\mathbf{x},\mathbf{o}))}{Z(\mathbf{x})}\pi(\mathbf{o}). \tag{6}$$

where $Z(\mathbf{x},\varepsilon)$ is a normalization (partition) function. Moreover, the Lagrange multiplier ε is determined by the value of the expected distortion, $\mathcal{D}$, is positive and satisfies, $\frac{\partial \mathcal{R}}{\partial \mathcal{D}} = -\varepsilon$.

We have that among all the conditional distributions, the one that optimizes this functional in Equation 5 is π^{ε} in Equation 6 above. This is our main result, that the distribution that minimizes the privacy risk, given a distortion constraint is a differentially-private distribution. From examining equation 1 and Theorem 1 we have

Theorem 4. *The distribution that minimizes Equation 4 defines a $2\varepsilon\Delta\,\mathbf{dist}$-differentially private mechanism.*

Figure 1 illustrates the tradeoff. It plots the unconstrained Lagrangian function $L(\mathcal{D}, \mathcal{R}) = \mathcal{D} + \frac{1}{\varepsilon}\mathcal{R}$, which because of the convexity of the risk-distortion function is also convex. For a given privacy parameter ε, we consider lines of slope $-\varepsilon$. We see that these lines intersect the curve at various points, these points represent the risk-distortion tradeoffs for those values. As we should expect, a high privacy-risk implies a low distortion and vice-versa. We see that for a given value of $-\varepsilon$, the line that is tangent to the curve at represents the optimal tradeoff point between the risk and the distortion. The value of the function $L(\mathcal{D}, \mathcal{R})$ on these lines is a constant, which implies that in some way the level of privacy imposes a value on the function L, since such a line can only intersect the curve in at most two places.

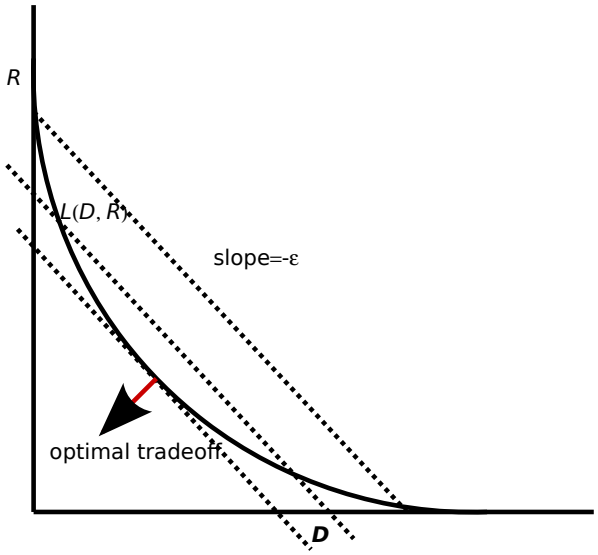


Fig. 1. Risk-distortion curve

4 Differential Privacy Arising Out of the Maximum Entropy Principle or Minimum Discrimination Information Principle

The *principle of maximum entropy* was proposed by Jaynes [7]. Suppose, a random variable $\mathbf{X}$ takes a discrete set of values $\mathbf{x}_i$ with probabilities specified by $p_{\mathbf{X}}(\mathbf{x}_i)$, and we know of constraints on the distribution $p_{\mathbf{X}}$, in the form of expectations of some functions of these random variables. Then the principle of maximum entropy states that of all distributions $p_{\mathbf{X}}$ that satisfy the constraints, one should choose the one with the largest entropy $H(\mathbf{X}) = -\sum_i p(\mathbf{x}_i) \log(p(\mathbf{x}_i))$.

In the case of a continuous random variable, the Shannon entropy is not useful and for such cases we apply the principle of minimum discrimination

information [7]. It states that given a prior p on $\mathbf{X}$, a new distribution q should be chosen so that it as hard as possible to distinguish it from the prior distribution p, that is the new data should produce as small a gain in information as possible given by $D_{KL}\left(q\|p\right)$.

We show that the application of the principle of Maximum Entropy to the distribution $\pi(\mathbf{o}|\mathbf{x})$ gives rise to a differentially-private mechanism.

When trying to find a distribution $\pi_{\mathbf{O}|\mathbf{X}}(\mathbf{o}|\mathbf{x})$, we utilize the Maximum Entropy Principle. Among all distributions $p(\mathbf{o}|\mathbf{x})$, we choose the one that maximizes the entropy $H(\mathbf{O}|\mathbf{X})$ subject to satisfying the constraint that the expected distortion function $\mathbf{dist}(\mathbf{o},\mathbf{x})$ is bounded by a quantity D. So we have,

$$\textbf{maximize } H(\mathbf{O}|\mathbf{X})$$

$$\textbf{subject to } \sum \mathbf{dist}(\mathbf{x},\mathbf{o})p(\mathbf{o}|\mathbf{x})p(\mathbf{x}) \leq \mathcal{D}.$$

From equation 2 we observe that minimizing the mutual information as in Equation 4 is equivalent to maximizing the entropy $H(\mathbf{O}|\mathbf{X})$.

Shannon introduced the concept of *equivocation* as the conditional entropy of a private message given the observable [15]. Sankar et. al [14] use equivocation as a measure of privacy of their data transformation. Their aim is also to maximize the average equivocation of the underlying secret sample given the observables Since $I(\mathbf{X};\mathbf{O}) = H(\mathbf{X}|\mathbf{O}) - H(\mathbf{X})$, minimizing $I(\mathbf{X};\mathbf{O})$ is also equivalent to maximizing the conditional entropy $H(\mathbf{X}|\mathbf{O})$, subject to constraints on the expected distortion. Therefore, the exponential distribution $\pi^{\varepsilon}(\mathbf{o}|\mathbf{x})$ as defined in Equation 6 maximizes the conditional uncertainty about the underlying sample given a constraint on the distortion function.

Now consider the worst case which differential privacy protects against, that is given knowledge of the entire database except for one row i, represented as $\mathbf{X}_{-i}$, if we look at the problem of maximizing the uncertainty of the random variable $\mathbf{X}_i$, we have

$$\text{maximize } H(\mathbf{X}_i|\mathbf{O},\mathbf{X}_{-i})$$

$$\text{subject to } \sum \mathbf{dist}(x_i,\mathbf{x}_{-i},\mathbf{o})p(x_i|\mathbf{x}_{-i},\mathbf{o})p(\mathbf{x}_{-i},\mathbf{o}) \leq \mathcal{D}$$

Again this is equivalent to minimizing the mutual information $I(\mathbf{X},\mathbf{O})$ when $\mathbf{X}_{-i}$ and $\mathbf{O}$ are given.

A note on incorporating auxilliary information: Usually, differential privacy provides guarantees on the inference, irrespective of any side or auxilliary information. This can be easily incorporated in our framework like Sankar et. al [14] by making all the distributions above conditional on the side information.

5 Conclusion and Future Work

We presented an information-theoretic foundation for differential privacy, which to our knowledge is the first such attempt. We formulated differential privacy within the broader frameworks of various problems in information theory such

as the rate-distortion problem and the maximum entropy principle. There are several directions for future work.

One, we can try to apply the risk-distortion framework to examine the generation of private synthetic data when the underlying data generating distribution $p_{\mathbf{X}}(\mathbf{x})$ is known. Additionally, one could try derive bounds on the mutual information in such cases. Second, we can examine the deployment of this framework to problems where the distortion function **dist** is specified. Another direction is to examine the notion of compressive privacy [9] in this rate-distortion framework and derive bounds for the rate.

References

1. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Degano, P., Palamidessi, C.: Differential privacy: On the trade-off between utility and information leakage. In: FST 2011 (2011)
2. Barthe, G., Kopf, B.: Information-theoretic bounds for differentially private mechanisms. In: CSF 2011 (2011)
3. Bezzi, M.: An information theoretic approach for privacy metrics. TDP 2010 3(3), 199–215 (2010)
4. Cover, T.M., Thomas, J.A.: Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing), 2nd edn. Wiley-Interscience (July 2006)
5. Dwork, C.: Differential Privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006, Part II. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006)
6. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating Noise to Sensitivity in Private Data Analysis. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 265–284. Springer, Heidelberg (2006)
7. Jaynes, E.T.: Information theory and statistical mechanics. ii. Phys. Rev. 1957 108, 171–190 (1957)
8. Li, N., Li, T.: t-closeness: Privacy beyond k-anonymity and -diversity. In: ICDE 2007 (2007)
9. Li, Y.D., Zhang, Z., Winslett, M., Yang, Y.: Compressive mechanism: Utilizing sparse representation in differential privacy. CoRR, abs/1107.3350 (2011)
10. McGregor, A., Mironov, I., Pitassi, T., Reingold, O., Talwar, K., Vadhan, S.P.: The limits of two-party differential privacy. In: FOCS 2010 (2010)
11. Mcsherry, F., Talwar, K.: Mechanism design via differential privacy. In: FOCS 2007 (2007)
12. Mir, D.: Differentially-private learning and information theory. In: EDBT-ICDT-W 2012 (2012)
13. Rebollo-Monedero, D., Forne, J., Domingo-Ferrer, J.: From t-closeness-like privacy to postrandomization via information theory. IEEE TKDE 2010 22, 1623–1636 (2010)
14. Sankar, L., Rajagopalan, S., Poor, H.: A theory of utility and privacy of data sources. In: ISIT 2010 (2010)
15. Shannon, C.: Coding theorems for a discrete source with a fidelity criterion. IRE National Convention Record, Part 4, 142–163 (1959)
16. Tishby, N., Pereira, F.C., Bialek, W.: The information bottleneck method. In: Allerton 1999 (1999)
17. Vora, P.L.: An information-theoretic approach to inference attacks on random data perturbation and a related privacy measure. IEEE Trans. Inf. Theor. 2007 53(8), 2971–2977 (2007)

Author Index